USENIX

SYSTEMS ADMINISTRATION CONFERENCE (LISA VII) PROCEEDINGS

Autumn 1993

U
US
USE
U NIX
USENIX

CONFERENCE PROCEEDINGS

USENIX Systems Administration (LISA VII) Conference

November 1-5, 1993
Monterey, California

For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: 510-528-8649

The price is $25 for members and $33 for nonmembers.

Outside the U.S.A. and Canada, please add
$12 per copy for postage (via air printed matter).

ISBN 1-880446-56-1

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

Proceedings of the Seventh Systems Administration Conference (LISA VII)

November 1-5, 1993
Monterey, CA, USA

TABLE OF CONTENTS

Acknowledgments iv

Author Index v

Plenary Session

Wednesday (8:30-10:00) Chair: Bjorn Satdeva

Opening Remarks and Announcements
Bjorn Satdeva, /sys/admin, inc.

Keynote Address: A Management View of Systems Administration
John Black, Oracle

Vis Á Vis

Wednesday (10:30-12:00)

Collaborative Networked Communication: MUDs as Systems Tools 1
Rémy Evard, Northeastern University

Horses and Barn Doors: Evolution of Corporate Guidelines for Internet Usage 9
Sally Hambridge & Jeffrey C. Sedayao, Intel Corp.

Our Users Have Root! 17
Laura de Leon, Mike Rodriquez, & Brent Thompson, Hewlett-Packard Company

Software Installation and Customization

Wednesday (1:30-3:00)

LUDE: A Distributed Software Library 25
Michel Dagenais, Ecole Polytechnique de Montréal; Stéphane Boucher, Bell-Northern Research; Robert Gérin-Lajoie, Universite de Montréal; Pierre Laplante, Centre de Recherche Informatique de Montréal; Pierre Mailhot, Universite de Montréal

The Corporate Software Bank 33
Steven W. Lodin, Delco Electronics Corporation

Customization in a UNIX Computing Environment 43
Craig E. Wills, Kirstin Cadwell, & William Marrs, Worcester Polytechnic Institute

Source Control

Wednesday (3:30-5:00)

Local Disk Depot – Customizing the Software Environment 51
Walter C. Wong, Carnegie Mellon University

Methods for Maintaining One Source Tree in a Heterogeneous Environment 57
Bjorn Satdeva, /sys/admin, inc.

Towards a POSIX Standard for Software Administration 67
Barrie Archer, ICL

Getting Down To Busines

Thursday (8:30-10:00)

Managing the Mission Critical Environment 81
E. Scott Menter, Enterprise Systems Management Corporation

Open Systems Formal Evaluation Process 87
Brian William Keves, Systems And Network Management

A Case Study on Moves and Mergers 93
John Schimmel, Silicon Graphics Inc.

System Administration by Delegation

Thursday (10:30-12:00)

Guerrilla System Administration: Scaling Small Group Systems Administration To a Larger Installed Base 99
Tim Hunter & Scott Watanabe, University of Colorado, Boulder

Role-based System Administration or Who, What, Where, and How 107
Dinah McNutt, Tivoli Systems

Delegation: Uniformity in Heterogeneous Distributed Administration 113
Jean-Charles Grégoire, INRS-Télécommunications

System Administration Tools

Thursday (1:30-3:00)

satool – A System Administrator's Cockpit, An Implementation 119
Todd Miller, Christopher Stirlen, Evi Nemeth, University of Colorado, Boulder

Sysctl: A Distributed System Control Package 131
Salvatore DeSimone & Christine Lombardi, Project Agora, IBM T. J. Watson Research Center

Automated System Monitoring and Notification with Swatch 145
Stephen E. Hansen & E. Todd Atkins, Stanford University

Disk Management

Thursday (3:30-5:00)

Where Did All The Bytes Go? 157
Dinah McNutt, Tivoli Systems; Michael Pearlman, Rice University

A Practical Approach to NFS Response Time Monitoring 165
Gary L. Schaps & Peter Bishop, Cirrus Logic, Inc.

The Amanda Network Backup Manager 171
James da Silva & Ólafur Guðmundsson, Department of Computer Science, University of Maryland

Information Tools

Friday (8:30-10:00)

PLOD: Keep Track of What You're Doing 183
Hal Pomeranz, QMS, Inc.

How to Keep Track of Your Network Configuration 189
J. Schönwälder & H. Langendörfer, TU Braunschweig, Germany

Forecasting Disk Resource Requirements for a Usenet Server 195
Karl L. Swartz, Stanford Linear Accelerator Center

The Human Behind The Root

Friday (10:30-12:00)

Simplifying System Administration Tasks: The UAMS Approach 203
Roland J. Stolfa, Oklahoma State University

System Administration as a User Interface: An Extended Metaphor 209
Wilson H. Bent, Jr., University of Southern California

The System Administration Maturity Model – SAMM 213
Carol Kubicki, Motorola Cellular Infrastructure Group

Potpourri

Friday (1:30-3:00)

Establishing and Administering a Public Access Internet facility 227
Sheri Byrne, Gemini Learning Center

Implementing Execution Controls in Unix 237
Todd Gamble, WilTel Network Services

HLFSD: Delivering Email to Your $HOME 243
Erez Zadok, Columbia University; Alexander Dupuy, System Management ARTS

ACKNOWLEDGMENTS

GENERAL CHAIR
Bjorn Satdeva, */sys/admin, inc.*

PROGRAM COMMITTEE
Brent Chapman, *Great Circle Associates*
Lee Daemon, *Castle PAUS*
Tina M. Darmohray, *Lawrence Livermore National Labs*
Janet Frazer, *UNIX System Laboratories*
Helen Harrison, *SAS Institute*
Dinah McNutt, *Tivoli Systems*
Bryan McDonald, *SRI International*
Arch Mott, *cisco Systems, Inc.*
Jeff Polk, *Berkeley Software Design, Inc.*
Greg Rose, *Australian Computing & Comm. Inst.*
Peg Schafer, *Bolt, Beranek, & Newman, Inc.*
Steve Simmons, *Inland Sea*
Lisa Y. Weissler, *RAND Corporation*
Pat Wilson, *Dartmouth College*
Elizabeth Zwicky, *SRI International*

TUTORIAL COORDINATOR
Daniel V. Klein, *USENIX Association*

TERMINAL ROOM COORDINATOR
Arch Mott, *cisco Systems, Inc.*

PROCEEDINGS PRODUCTION
Rob Kolstad, *BSDI*
Carolyn Carr, *USENIX*
Malloy Lithographing, Inc.

USENIX MEETING PLANNER
Judith F. Desharnais, *USENIX Association*

USENIX EXECUTIVE DIRECTOR
Ellie Young, *USENIX Association*

USENIX SUPPORT STAFF
Diane DeMartini, *USENIX Association*
Patrick Dunne, *USENIX Association*
Toni Veglia, *USENIX Association*

VENDOR DISPLAY COORDINATOR
Cynthia Deno, *USENIX Consultant*

AUTHOR INDEX

Barrie Archer 67
E. Todd Atkins 145
Wilson H. Bent, Jr. 209
Peter Bishop 165
Stéphane Boucher 25
Sheri Byrne 227
Kirstin Cadwell 43
Michel Dagenais 25
Salvatore DeSimone 131
Alexander Dupuy 243
Rémy Evard 1
Robert Gérin-Lajoie 25
Todd Gamble 237
Jean-Charles Grégoire 113
Ólafur Guðmundsson 171
Sally Hambridge 9
Stephen E. Hansen 145
Tim Hunter 99
Brian William Keves 87
Carol Kubicki 213
H. Langendörfer 189
Pierre Laplante 25
Laura de Leon 17
Steven W. Lodin 33
Christine Lombardi 131
Pierre Mailhot 25
William Marrs 43
Dinah McNutt 107
Dinah McNutt 157
E. Scott Menter 81
Todd Miller 119
Evi Nemeth 119
Michael Pearlman 157
Hal Pomeranz 183
Mike Rodriquez 17
Bjorn Satdeva 57
J. Schönwälder 189
Gary L. Schaps 165
John Schimmel 93
Jeffrey C. Sedayao 9
James da Silva 171
Christopher Stirlen 119
Roland J. Stolfa 203
Karl L. Swartz 195
Brent Thompson 17
Scott Watanabe 99
Craig E. Wills 43
Walter C. Wong 51
Erez Zadok 243

Collaborative Networked Communication: MUDs as Systems Tools

Rémy Evard – Northeastern University

ABSTRACT

A systems administration group is only as effective as its internal communication mechanisms. On-line communication has traditionally been managed via electronic mail or news, which are neither real-time nor truly collaborative. Communication tools which let multiple parties work together in real-time have become widespread on the Internet in the last several years.

In an effort to keep a physically disjoint systems staff working together effectively, we have explored the use of MUDs as communications tools. By allowing many people to interact in an extensible environment, MUDs have solved many of the problems that we had with on-line communication, and provided many unexpected benefits as well.

Introduction

Multi User Dungeons, or MUDs, are widely used on the Internet as interactive role-playing games. They use valuable network resources, attract unruly users, and are often run by students who didn't quite bother to ask the permission of the local authorities. As such, they are considered one of the banes of systems administrators, and perhaps rightly so.

Recently MUDs have been seen in a new light by some. While they are used most often as gaming environments, the software is in no way constrained to just that purpose. Instead, it is possible to program an environment in the MUD that is suitable for socializing and communicating. The MUD becomes a virtual ''place'' on the network where people can meet and collaborate on various projects.

In this paper, we present our experiences in using a MUD as a tool for improving the communications of our systems administration group.

Site Information

The Experimental Systems Group manages the computing environment within the College of Computer Science at Northeastern University, which is located in Boston, Massachusetts. The College operates approximately 300 computers of various types, including Macs, PCs, and a set of UNIX workstations consisting primarily of SPARCs and DECstations. About 800 users keep the computers, the network, and the systems group very busy.

The core group currently consists of five full-time staff members (two of whom are students enrolled in a cooperative education program) and several over-active student volunteers. In addition, eight to twelve students participate in systems-related projects each term of the school year as part of a volunteer program, and may put in as much time as the full-time staff (or perhaps more).

With this many people involved in systems projects, coordination and communication become essential to making effective progress. The work places are scattered around the building, many people work from home, and the students and hackers on the group tend to keep unconventional hours, so meeting physically in order to coordinate is not always practical.

Communication Needs of a Systems Group

In any organization, communication between its members dictates how well the group functions. This is true of a community of any size. Consider the havoc created when a telephone network ceases functioning, the problems attributed to lack of communication in today's families, or (sigh) what happens when electronic mail quits flowing.

Different communications tools are appropriate for different types of communication. The users on our network notify us of problems and make requests by sending electronic mail to ''systems''. We work with them individually either through email or in person. We use newsgroups for announcing changes which will affect everyone, such as impending downtime or new software installations. We also have newsgroups for open discussion of systems-related issues.

Internally, much of the systems group's coordination is done via electronic mail. (We have a separate alias that we don't share with the user community. This is enormously useful when combined with judicious use of mail filters.) Email within the group is nice for announcing future plans, sending notices about changes, or communicating nearly any non time-critical type of information. It provides a handy way to log changes, is extremely convenient, and is largely non-interruptive, so you may choose when to read your new email. (If you're on the group, you always have new mail.)

Despite its flexibility, we discovered that electronic mail was not appropriate for all of the types of communication that we needed. Time-dependent information is one example: sending email saying "Is anyone in the machine room right now?" just doesn't work if the person in the machine room doesn't read their mail while there. Further, it's not very effective for round-table discussions. While email can be used this way, we have noticed that the quoting mechanism commonly used in electronic mail can turn a potential brainstorming session into a nitpicking mechanism. Worse, it's not quite real-time, so the discussion will tend to die out, or become multi-threaded.

Because of the distributed nature of our group, we needed to be able to have on-line discussions and to communicate about real-time issues. Email wasn't working out well, and news clearly wasn't the right direction in which to move. Many programs available on the Internet provide interactive networked communication, and presented interesting possibilities. After some exploration, we decided to test a MUD for a few weeks.

MUDs

MUD stands for Multi User Dungeon (or, pretentiously, Multi User Dimension). The original MUD was written in 1979 by Roy Trubshaw and Richard Bartle on a DECsystem-10 at Essex University [Bart90]. It was a game similar to the classic Colossal Cave adventure, except that it allowed multiple people to play at the same time and interact with each other. Reportedly, this was enormously popular, and the idea caught on.

The original idea has evolved over the years into a client/server architecture. The MUD server manipulates the database of objects in the virtual world, is programmable in some sort of language that allows one to extend the set of objects, and accepts network connections from clients. The client's primary task is to send and receive I/O between the server and the user. The MUD server exists on one machine on the network, while the client is typically run by the users on their own machines.

Today, there are perhaps a dozen popular types of MUDs available on the Internet. They vary in many details, such as their embedded programming language and their storage methods for the objects they manipulate, but all have the capacity to allow multiple users to interact within some shared context.

Interacting with a MUD

The majority of existing MUDs are text-based. One types commands to them using a primitive English-like set of commands, and sees the results in a like manner. They are very reminiscent of the original Colossal Cave adventure.

For example, I could type:

```
look
```

and the response might be:

```
You are standing in a sunny, grassy
park-like area, under the sprawling
limbs of a weeping willow tree,
growing on the banks of a small
stream.  To the south you see a
university campus.  A trail winds
north into a forest.  Erik is here.
```

Technically, what happened is that I typed the string "look" to my client program. It sent that command to the MUD server, which parsed it (not very difficult in this case). The server then calculated the effects of my command on the objects in the server (which was to retrieve the description of my location), and sent the result back to my client, which printed it on my screen.

In order to establish a connection with a MUD, you must have a "character" on the server. You must supply a password to login to the MUD as that character, much like when you login to a UNIX workstation. Once the connection has been opened, all the commands that you type are perceived to come from your character. When you close the connection, the state of your character (location, possessions, etc) may or may not be preserved by the server.

The MUD server typically presents a virtual space organized into "rooms". A room, in the MUD sense, corresponds to a place where characters or objects may be located. In the example above, I was in a room described as a park-like area. If I typed "north", I would have been moved to a new room, presumably some ways down the windy path and into the forest.

The primary means of communication within a MUD is by talking to other people who are located in the same room as you are. An example transcript might look like:

```
Woj [to Remy]: I looked in Tom's
     office, and found that we don't
     have the right kind of scsi
     cable.
> say Hmm.
You say, "Hmm."
Ivan says, "We've got to have one
    somewhere."
```

```
Woj says, "Well, we do have one,
    but it's only a half-foot
    long."
> emote tries to think of the
    type of cable you need.
Remy tries to think of the type of
    cable you need.
Woj says, "Or, I could turn off
    alewife and steal that one..."
> say "Check my office, if there's
    not one there, bounce alewife."
You say, "Check my office, if
    there's not one there, bounce
    alewife."
Rob says, "Anyone up for some ice
    cream?"
Brian <-
Woj says, "Wait, I'm on my way!"
> emote sighs and chuckles.
Remy sighs and chuckles.
```

A character may say something using the `say` *`something`* command. They may indicate some sort of action by using the `emote` *`something`* command. I typed `emote sighs and chuckles.` in the example above. All of these common commands have short cuts to make them easy to type (`say` can be shortened to a quotation mark, while `emote` can be shorted to a colon) and come naturally after a few minutes of trying them.

It is possible to talk privately with a person using some form of a whisper command, or to talk to someone who is not in the same room by using a paging command. Many other commands exist, and may be created as needed.

One interesting aspect of MUDs is that they impose a spatial metaphor on the participants. One may talk and interact easily with people in the same virtual room, and may use other means to communicate with people in other locations. In this way, the MUD becomes a virtual space reachable via the network.

Applicability of MUDs to Systems Communication

We felt that MUDs had several features that would make them a useful communications tool for the systems group:

- MUDs are interactive in real-time. When one says something on the MUD, all the intended recipients see it immediately. They can answer in the amount of time it takes to type their response.
- MUDs are a networked service. Clients and servers simply need to be on the same network in order to connect to each other. Thus one need not be logged into the same computer as the people with whom one is communicating.
- MUDs are multiuser capable. A large number of people can interact with each other at once.
- MUDs are extensible. Any decent MUD server will have an embedded programming language that may be used to extend the database of server objects and to create new commands. If the tool is to be adapted to new uses, it must be flexible.
- MUDs are exclusive. Only people who have been given characters on the MUD are allowed to connect to it. This is a crucial feature. Any of the 800 users on our network can send me email, but only the systems group can talk to me on the MUD.
- MUDs, in conjunction with most clients, have a history mechanism. Even though the interaction on a MUD happens in real-time, it can be recorded by a client that is connected. Thus one may read a conversation that happened when one wasn't actively involved. Clients also have the ability to save transcripts to files, allowing for a permanent archive of important communication.

We explored other communications tools as possible alternatives. A summary of our findings and opinions is presented in the appendix.

Other Interesting Aspects of MUDs

Once we started using a MUD as a tool, we discovered several other useful features that we had not considered previously.

As mentioned above, MUDs provide a metaphor of real life (or virtual life). In a MUD, people and things exist in a place. One interacts with an object as one would in real life. This creates an interesting context for solving problems and indicating real-life situations. For example, in the systems MUD, I have an attic which is located above the cabin where we normally work. When I'm very busy in reality, I move to that attic, and leave everyone else in the cabin. In this way, I am out of the flow of conversation, but still available on the MUD if someone wishes to discuss a specific issue with me. The spatial metaphor is also used when we build systems tools into the MUD.

Because a MUD is extensible via a programming language, the objects in the MUD can be programmed to do anything that you could do with any other type of program. This is an extremely powerful tool, and we are just beginning to make use of it. For example, we have considered constructing a version of a new building that the College may be expanding into, and then simulating the physical network connectivity within the MUD. We will do this before the building is built in reality, in order to foresee problems that may crop up.

One tool that currently exists in our MUD is a Gopher client. We ported this code from another

MUD, where it was originally developed [Masi93]. Instead of simply running a command that pops a Gopher interface up on the MUD, the Gopher client was built into the spatial metaphor of the MUD, and exists in the MUD as a "Gopher slate", something like a portable computer. People in the MUD may make their own Gopher slates and manipulate them by navigating Gopher menus, much like one does with a conventional Gopher client. Gopher slates differ from other clients in that they may be shared with other participants in the MUD – one can bring up a document on a Gopher slate and then hand it to someone so that they too can read it, or they can put it on a table and jointly manipulate it. The MUD has then become a way in which we can collaborate in our use of Internet resources.

Other items in the MUD that have been programmed are a dictionary, a weather map, and various systems status monitors.

Running a MUD

We are running a type of MUD known as a MOO, which stands for MUD, Object-Oriented. A MOO keeps the database of objects in memory, which means that the process tends to be large. Other types of MUDs keep unreferenced objects on disk, and therefore take up less in-core memory. We chose to use a MOO because of possible future research directions; had we been concerned about computing resources, it would have been wiser to choose a different type of MUD server.

MUDs can be run on practically any UNIX machine; we're running our MOO on a SPARC 670/MP. It currently takes 9 megabytes of disk space to store the code and several backup copies of the object database. The process takes anywhere from 5 to 15 megabytes of memory, depending on what is happening within the MOO. We have found that the server does not create an appreciable load on the computer, and could be run unobtrusively on a slower machine without any trouble; we used the SPARC simply because it was available when we started the MUD.

The MUD client runs on whatever machine the user is using. It is possible to use **telnet** to connect to the MUD. Telnet is not the most useful client, however, because it doesn't separate input from output, and the output from the MUD tends to get confused with the command that one is trying to type. Specialized clients that are easier to use have been written for for the X Window System, emacs, and for several types of personal computers.

For the most part, the members of the systems group use a client for emacs named **mud.el**, which connects to a MUD within an emacs buffer. This allows one to edit commands before sending them to the MUD, to capture output into another buffer, and to keep a history of everything that happened in the MUD that may be viewed as "scrollback" at a later date. Using emacs also makes it convenient to edit code for objects within the MUD itself.

Typically, a member of the group will start up a small-sized emacs on their workstation, connect to the MUD, and then leave it going for the whole time that they are logged in, occasionally glancing at it to see what's happening. Some people use the **screen** program to start an session that may be moved from one controlling terminal to another; this allows one to login to the workstation from another location (from home, for example), without disconnecting from the MUD. In this way, one may read what happened while one was absent, and keep up with any happenings. Other people prefer to disconnect completely when they're not going to be around.

Experiences and Observations

The original environment that we built was a replica of the building that we work in. Everyone built their office, we implemented a network that one could travel through, and we constructed a central meeting place. But we found that having one real-life version of our urban university was enough. Instead, we created a virtual space that is different and somewhat more pleasant, and this has seemed to change the mood of the MUD for the better.

We have named the MUD environment that we built ''The InfoPark''. The InfoPark is in an outdoor setting, with trees, streams, and animals. In the middle of a forest is a cabin, where most members of the systems group go when they're connected. We've also built a university campus for use by one of the research groups within the College, where they plan to hold on-line meetings.

We have found that the MUD is an effective way to hold pre-arranged meetings for people who can't be in the same physical location. We have had virtual systems meetings three or four times, each about various topics. We save a transcript of the meeting and email it to people who weren't present, and refer to it when trying to remember exactly what issues had been raised. Using a MUD in this way is not as time-effective as meeting in reality, but is at least as useful as having a conference telephone call.

We use the MUD as a coordination mechanism. People tend to announce on the MUD what they are doing in real life. Phrases like ''Jim heads into the machine room to check the tapes'', ''Ivan is about to reboot amber'', or ''I'm hungry, who's interested in lunch?'' are commonly seen. We have found this to be so useful that we have encouraged it by writing utilities that people can use to indicate why they are ''idling'' in the MUD. A character is called ''idle'' when they do not respond to activities within the MUD. This normally happens because the user has quit paying attention to the MUD for some reason. When a character is idling ''for office visitors'' or ''to drive home'', the other participants in the MUD can look at that character and see why it idled. In this way, we use the text-based virtual reality to reflect what is happening in real life. Before the MUD, we saw each other only at meetings, or after running all over the building trying to locate each other.

The MUD is used as a quick brain-storming or problem-solving mechanism, often in response to a recent email message. It's common for us to have a five-minute conversation on the MUD about a small systems issue, such as some detail of DNS, how to implement an extension to the file system, or how to fix some user's problem. Previously, these conversations would have happened through slower email, through office visits, or at regular systems meetings. All of these mechanisms are more cumbersome, and would have happened much less frequently. Thus the MUD has enabled new communications patterns.

The largest unexpected benefit of the MUD is that it created a social environment that didn't exist. People have real conversations on the MUD with other participants. These are typically, but not always, about systems-related issues. Because of this, members of the group find out about projects that they're not involved in. The new undergraduate volunteers come to know senior members of the staff that they would otherwise not recognize. The MUD has become a social place for the systems group that is populated even when everyone is logged in from home. It may be said that this takes away from work hours, or cuts down on effectivity while in the office. But, over time, we have seen our systems group, people who had simply worked on related jobs, grow into a real team. This change is due partly to our shared social context.

It is common for someone on the MUD to have a real-life interruption, be it a phone call, an office visitor, or simply being too busy to pay any attention to the MUD. Thus people tend to become inactive suddenly on the MUD. This doesn't create a communications problem – whenever they resume, the buffer is there, containing the old conversation, and it can be continued without difficulty. The problem it presents is when one needs to find a specific person who is on the MUD but isn't paying any attention to it. Our solution is to use a paging verb that causes the idle person's terminal to make an audible beep. If that person can, they will respond to the MUD. If they can't, they probably aren't anywhere near a computer. This is when communicator badges would come in handy (or be horribly annoying, depending on your point of view).

Many of the undergraduates who work with the group have cooperative education jobs in other parts of Boston. During the day, or even for a whole summer, they are unable to come into Northeastern. Because they stay on the mailing lists and can continue to interact with us on the MUD, they are constantly in touch, and can continue to work on projects. It is often difficult to tell whether a person is actually in the building (without asking them, that is), and has become increasingly less important over time as our network communication tools improve.

As part of the experiment, we have invited systems administrators from another site to join us on our MUD. This has been interesting – they often have unique viewpoints on how to solve our problems. We have traded information about tools, discussed collaborative projects, and are more in touch with what is happening at their site. This has been successful largely because we had already had professional relations with them. We originally thought that they should have a separate room in the MUD to discuss their own problems, but it hasn't really been used much. We would have to say that the collaboration between the two groups has been a success, but not a huge success. Perhaps the most interesting thing to come out of it is that they are building their own MUD.

We have created a bulletin board in the MUD that has the list of current systems projects. This has been a failure because it's not really linked in with our other mechanisms for project organization. Once these are mechanisms are more clearly defined, we will undoubtedly build a MUD interface in order to see whether or not it will be useful.

Problems

While the MUD solves many problems, it also has a few of its own. First, as mentioned above, it can be easily interrupted by something happening in real life. This is something that one simply gets used to.

When many people are in a room, the conversations can get confused and intertwined. One quickly learns to pay attention to the conversation one is involved in, and to partially ignore the others. The use of recipient indications (such as ''Rémy [to Ivan]: Where are you?'') solves a lot of the problem. This is, again, something that one adapts to very quickly. On the other hand, it doesn't scale very well. We considered opening our MUD up to anyone who read this paper, but we don't yet understand how to coordinate a hundred people talking to each other.

The MUD requires a bit of time to learn. There are perhaps 10 commands that everyone must learn immediately (say, emote, page, whisper, look, and so on). Learning to administer a MUD is more difficult. This involves making sure the database is backed up, creating or disabling guest characters, and learning the MUD's programming language in order to extend the environment. This is something that can be done incrementally, but does take time. On the other hand, it can also be fun.

Perhaps the biggest problem we've discovered is that the MUD has the potential to be a big distraction. When it's constantly running somewhere on your screen, it can interrupt your concentration. Everyone learns to deal with this differently. I have simply gotten used to the flow of text across one of my windows, and tend to glance at it every few minutes to see what's happening. If I'm busy, I may not look at it for hours. If something important comes up, someone will page me. Several other people on the group iconify the window or bury the buffer below another one, or, when very busy, disconnect. This is part of a bigger problem of time management, and is related to similar interruptions caused by telephones, electronic mail, and office visitors. We have not found the MUD to be any worse than the other distractions.

Future Work

We plan on several possible avenues for future growth. One project that is currently underway are objects that represent entities on the network. We'd like a printer in the MUD that always shows the various printer queues on the network, and has the ability to restart printer daemons (with appropriate security features built in). We're working on a themely representation for computers that will notify us when they're unreachable. Perhaps:

```
A worried-looking squirrel scampers
in through the window and says,
"denali is unreachable at 3:45 pm"
```

... then again, perhaps not.

Current research at Xerox PARC involves integrating MUDs with audio, video, and shared programs [Curt93]. The MUD then becomes the perfect sequencer for these devices because of its spatial metaphor: one could actually talk with and see the people who are in the same room on the MUD, as opposed to hearing everything that is said on a simple shared audio channel (which is what primarily happens with today's multicast backbone interfaces). One could look at a white-board on the wall of the cabin and it have it pop up as an X Windows application on the screen. The application would be shared by anyone looking at the board, and changes could be made and seen by everyone. We expect these tools to be enormously useful, but are concerned that the audio and video may not be as useful as one might think, given the interruptive capability they have, and the lack of a history mechanism.

As other people on other MOOs write neat applications, we will be porting those onto our MUD. Shared World Wide Web readers are likely, as are other interfaces to the Internet's resources.

We are working on a way to access systems administration information from within the MUD, in order to make it even more of a collaborative environment where administrators can meet and discuss problems.

Because of the value of the MUD as a place for virtual meetings, we have created a conference room that we will be sharing with faculty and various groups within the University. We had thought of allowing students on the MUD in order to learn some aspects of object oriented programming, but we value our peace and quiet and will be letting them run their own MUD, where they won't be able to find us.

Conclusion

Our MUD has been a successful experiment, and we plan on continuing to use it. It has solved the communications problems we had, and has provided some unexpected benefits. Among those are the creation of communication channels that had not previous existed, and the ability to work more effectively from remote locations. We recommend it to other systems administration groups and are interested in hearing about their experiences.

Availability

The MOO server is available on parcftp.xerox.com in `/pub/MOO`. Get the latest version of the LambdaMOO code. It runs on nearly any type of UNIX platform. Clients, other MOO code (such as the Gopher slates), and related papers are also at this site.

LP, another type of MUD, is available on ftp.ccs.neu.edu in `/pub/mud/drivers/lpmud`. Database libraries are available in the `/pub/mud/mudlibs` directory. Clients and papers are also on this ftp server.

Acknowledgements

Thanks to Robert Leslie for bringing up the first systems MUD; Erik Ostrom, Larry Masinter, Jay Carlson, and Michele Evard for writing a lot of the code which was ported to the InfoPark; Stephen White and Pavel Curtis for writing MOO; the Argonne National Laboratory Mathematics and Computer Science Support Staff for participating; and the whole InfoPark crew for (on-line and interactive) comments on this paper.

Author Information

Rémy Evard is the leader of the Experimental Systems Group of the College of Computer Science at Northeastern University. He has been at Northeastern for a busy year, during which he changed nearly everything about the network. He received his M.S. in Computer Science from the University of Oregon in 1992, and graduated from Andrews University with a B.S. in Mathematics and Computer Science in 1990. He has been doing UNIX systems administration for five years, but has only been MUDding for nine months. He may be reached as "remy@ccs.neu.edu", or as "r'm" on any of several MUDs.

Bibliography

[Bart90] R. Bartle, "Interactive Multi-User Computer Games", December 1990. ftp://ftp.ccs.neu.edu//pub/mud/docs/papers/mudreport.ps.gz

[Curt92a] P. Curtis, "LambdaMOO Programmer's Manual", August 1993. ftp://parcftp.xerox.com//pub/MOO/ProgrammersManual.ps

[Curt92b] P. Curtis, "Mudding: Social Phenomena in Text-Based Virtual Realities", in the Proceedings of the 1992 Conference on the Directions and Implications of Advanced Computing, Berkeley, May 1992. ftp://parcftp.xerox.com//pub/MOO/papers/DIAC92.ps

[Curt93] P. Curtis and D. Nichols, "MUDs Grow Up: Social Virtual Reality in the Real World", unpublished report, May 5, 1993. ftp://parcftp.xerox.com//pub/MOO/papers/MUDsGrowUp.ps

[Eich88] M. Eichin, R. French, D. Jedlinsky, J. Kohl, W. Sommerfeld, "The Zephyr Notification Server", in Winter 1988 Usenix Proceedings. ftp://athena-dist.mit.edu//pub/zephyr/doc/usenix.ps

[Masi93] L. Masinter and E. Ostrom, "Collaborative Information Retrieval: Gopher from MOO", in The Proceedings of INET '93, June 1993. ftp://parcftp.xerox.com//pub/MOO/papers/MOOGopher.ps

[Vers91] B. Verser, "Network utilization by MUD players (was Re: Gaming)," in Computers and Academic Freedom Newsletter, v.1 n.43, Dec 15, 1991. gopher://gopher.eff.org/1/academic/news/cafv01n43

Appendix – A Comparison of Communication Tools

Before the systems group decided to use a MUD, we considered several other tools that had various features. We present them, along with our own opinions, in the hopes that they will clarify the differences between various communications methods. A chart presenting our opinions is given in Figure 1. This is not intended to be an exhaustive summary, but more of an overview of different types of communication tools.

Asynchronous on-line communication tools

- Electronic mail is used to send a document to any number of recipients. It is, for the most part, reliable, and is used so constantly that it has become convenient through force of habit, if nothing else. It can either be interruptive or not, depending on the receiver's preference. While email communication can be so fast as to be nearly real-time, it does not normally convey the feeling of a real-life conversation. It is possible to keep a history of email messages. Email is probably the most useful communications tool a systems administrator has.

	Real Time	Online	Archivable	Unobtrusive	Multiuser	Exclusive	Extensible
Electronic Mail		●	●	●	●		
News		●	●	●	●		
write	●	●					
msend	●	●					
talk	●	●			●		
IRC	●	●	●	●	●	?	
Zephyr	●	●	?	?	●		●
MUDs	●	●	●	●	●	●	●
telephones	●						
pagers							
walkie-talkies	●				●		
communicators	●				●	●	

Figure 1: Communications Tools – Criteria and Opinions

- News (or USENET news), is a system that allows a document to be spooled on a machine for some time, where many people can access it. News can be shared across many machines or isolated to one site. It is good way to communicate with many people about various topics in a non time-critical manner. News is perhaps the best way to send announcements that many people should see. Multi-party discussions can work relatively well in news, but can take time, and have a tendency to diverge into irrelevant topics. News is normally accessed by the readers if and when they choose to read it, not when the news is first available.

Real-time on-line communication tools

- The UNIX **write** command is used to send a message to a user's terminal. It can be disabled by the receiver with the use of the **mesg** command. **Write** is real-time, but is inconvenient for all but the shortest of messages, because it only reads standard input and tends to create a mess on the receiver's screen.
- The **msend** utility is the next step up from **write**, allowing one to send messages to users of another computer. The input stream is more flexible, but the output still tends to clutter up the receiver's terminal.

Interactive on-line communications

- The UNIX **talk** command is useful for short conversations between two people. Enhanced versions of it allow conversations between any number of people. It is as real-time as it is possible to get over an ASCII connection, allowing one to see the typos made by the other person. We found talk to be annoying primarily because it is extremely interruptive (drat, which window is that beep coming from?), there is a confusion between talk protocols on various operating systems, and the interface is inflexible and primitive. Nonetheless we do use it occasionally.
- Internet Relay Chat, or **irc**, is a global chat program. It is useful for interactive real-time conversations between multiple people. While it would have solved most of our initial needs, we chose not to use it because of code stability problems, and because we weren't sure that it would truly be exclusive to the systems group.
- Zephyr, the communications server from Project Athena, provides a networked, scalable messaging service that is accessible via many types of clients. It allows real-time messages to be sent from one person to any number of people who are on the local network (for various definitions of local). With the right client it's possible to use Zephyr to have an ongoing conversation. More importantly, it can locate a user anywhere on the network and deliver the message (a feature that a MUD can not easily replicate). We think Zephyr may be as useful as a MUD and will be testing it in the future, perhaps in conjunction with MUDs. We did not initially try it because of the difficulty of bringing it up in a non-Athena environment.
- MUDs provide a multiuser extensible environment that can be used as a communications tool. A user connects to it by using telnet or a more sophisticated client. The primary disadvantages of a MUD are that it must be actively administered and that the user must initiate the connection. This is as opposed to Zephyr or email, where the user is located wherever they are logged in. These negatives did not significantly hinder us, so we elected to use a MUD.

Interactive off-line communications

- Telephones, pagers, cellular phones, and walkie-talkies are all solutions we've seen used. They have their uses when one is away from a computer, but are prohibitively interruptive. While they do solve the real-time communications problem, we didn't feel that provided the type of solution that we were looking for, largely because of the inconveniences that come with them. For example, telephones are only effective in real-time if one is near them when they ring, while walkie-talkies must be carried everywhere, and are constantly making noise. We originally thought that Star Trek communications badges would be exactly what we wanted but have since decided that they would simply be more advanced forms of annoyance devices. Perhaps the ideal solution for all of these will be the light-weight, mobile, internetworked, portable communications tool that may be turned off and ignored when necessary. A Powerbook running PPP over a cellular modem could provide interesting possibilities, for a price.

Looking at all the options, we felt that the combination of electronic mail and MUDs solved nearly all of our internal communications needs. Neither completely solves the problem, but they work as nice complements to each other. We use email primarily for internal announcements and instructions, and the MUD primarily for on-line coordination and planning.

Horses and Barn Doors: Evolution of Corporate Guidelines for Internet Usage

Sally Hambridge & Jeffrey C. Sedayao – Intel Corp.

ABSTRACT

Intel's Internet usage policy evolved from practically non-existent to explicitly defined – all in reaction to changing conditions and security threats. This paper covers the evolution of Intel Internet access policy, a continual struggle to close the barn doors before the horses get out. Throughout the paper, we outline key lessons we have learned during the policy-making process. It discusses Intel's first taste of the Internet, Intel's policy-making process, the open access policy of that period, and the resulting security challenges. It then covers the imposition of a stricter policy and implementing a firewall to enforce that policy. The paper proceeds to describe today's problems, the majority of which center around Intel people accessing the Internet. In response to this problem and growing numbers of people wanting to use the Internet, Intel has drawn up explicit corporate guidelines on Internet use. These guidelines are then compared to various Acceptable Use Policies and Netiquette guides. The paper concludes with some additional tasks Intel is planning in order to keep the barn doors closed.

Intel's Introduction to the Internet

Intel Corporation has had access to the Internet since 1987. At that time, we had a dial-up connection to the now defunct CSNET. We dialed Boston from Santa Clara, California several times a day to pick up and drop off mail. We did not have any kind of Internet access policy. We felt secure in having complete copies of all messages sent in and out and having our modems block dial-ins.

While the dial-up connection provided much-needed mail access to and from customers, vendors, and research partners, functionality was too limited. Delivery was so slow at times (days!) that paper proved a quicker and more reliable communication medium. Users complained that carrier pigeons would deliver mail faster. The long distance calls grew to be expensive. Because of these concerns and the desire for direct FTP and telnet access to the Internet, in 1989 we traded our CSNET dial-up connection for one with direct IP access over a leased line. An increase in functionality always means an increase in risk, as we will see in the next section.

The Challenges of an Open Door

Our first policy was this: anyone in the company could go out on the Internet, and rlogin, telnet and FTP access into Intel would be blocked. WE were the access providers, and so we imposed this policy unilaterally. The only place this was written down was in the router access list configuration.

What were the results of our (wide) open door? We received many complaints about Internet access from various system administrators around the company. They did not like the gaping door. Later, with unsolicited help from federal agents, we found some crackers who did.

- Key Lesson #1 – Research Policy Issues
- Key Lesson #2 – Consult with users and stakeholders on policy decisions
- Key Lesson #3 – Make the policy available and readable.

Our policy was incredibly naive. We did not think it through in depth and did not realize how easy it would be for intruders to exploit gaping holes. Furthermore, we did not have buy-in to our policy. System administrators weren't comfortable with it. Even worse, they were uncomfortable with a policy they couldn't even read. Things had to change.

Shutting the Door Part Way

The problems we encountered forced us to realize our mistakes. We looked into Internet access schemes implemented at other companies. We wrote down and proposed a limited access policy. This document was circulated for comment by electronic mail and presented at various user forums within Intel. Finally, we had the policy approved by an internal change control group. This was an official stamp that gave us legitimacy.

Our new policy restricted outbound Internet access to specific systems. Inbound access was limited to certain protocols going to dedicated servers. The outbound systems, controlled by site administrators, would be tightly controlled. Applications for Internet access systems would have to be signed by site network managers, the system administrator's manager, and our internal Information Security group. Applicants promised to read and obey our policy, which was circulated with the application forms.

- Key Lesson #4 – Get key people to buy into a policy. Better yet, get some kind of official stamp of approval.
- Key Lesson #5 – Forms with signature loops are a way of making sure that people are serious about wanting something. It is also a way to inform key parties of change and get their buy-in.

We managed to get people involved in making our policy. They bought into it, and we got an official stamp of approval from a internal group. By using forms, we weeded out people who weren't serious about managing Internet access systems. Moreover, we gave our Information Security group a chance to review and buy into the decision of who would want access.

- Key Lesson #6 – Provide metrics on usage and quality of service.

We made the decision that we would track how much the gateway was used and who was using it. We look at sheer volume, such as how many bytes each access system exchanges with the Internet and how many messages are exchanged through the gateway mail servers. We also decided to track some service metrics like mail delay through the gateway. An Internet gateway status and usage report is produced and widely distributed every quarter.

Keeping metrics has proven to be a good decision. We can track utilization, which helps us with capacity planning and with justifying new equipment. Management, initially unsure about funding our gateway, is usually persuaded when they see how much their people are using the Internet. Finally, keeping metrics gives us some idea how well we are managing the gateway.

Ironically, by shutting the door part way, usage boomed. Throughout the six years we have had mail capability, we have witnessed an exponential growth in the amount of mail coming into and going out of the company. This growth is consistent with Internet growth trends industry wide. (See Figures 1 and 2.)[1] Since Intel is a multi-site, multinational operation, almost all Intel sites dedicated a number of machines to provide ftp and telnet capability for groups within the site.

With growth in the number of Internet knowledgeable employees, (as well as those who have heard of the Internet but know little) we've seen demands for accounts on these machines skyrocket. We've also seen a corresponding growth in different kind of security problems – from Intel instead of to Intel. Most of these problems stem from people attempting logins to defunct accounts, or naively trying to telnet to ftp machines and vice versa. Still, even these innocent mistakes mean time and trouble. This is time and trouble for the system manager of the machine where the "break-in" is attempted as well as Intel's Internet contact and the system administrator of the internal Intel machine from which the "attempt" occurred. Intel personnel must then check system logs to determine who was logged in at the time, then contact those people to find out whether intent was indeed malicious. All of this takes time from resources which function better as network and system managers than High School Vice Principals.

We discovered that almost all of our policy focused on system and network administrators and not on users. Although we put conditions on how the access systems should be administered, we did not provide any tools or help to do so. We should not have been surprised that some of the Internet access systems were far more open than we liked. The incidents with misguided users sparked another fear. We could conceive scenarios [2] where a user could create an incident severe enough to cause Intel to shut down or tremendously restrict our Internet connection.

Getting the Horses to Behave

To combat these problems, an Internet Security Task Force was formed. This ad hoc group consists of representatives from Corporate Information Security and system managers and users. We had learned from past experience that only by getting people involved could we create workable policies.

Corporate Information Security bears the responsibility of protecting Intel's intellectual property assets. This group sets policy and procedures for Information Security, publishes a yearly summary of those policies, and has recently developed a class on information security for Intel employees.

In its Internet Policies, the Task Force has tried to maintain a balance between getting people to information (and information to people) and maintaining reasonable security. First, although most of us eschew bureaucracy, we ask those users requesting accounts on machines which have Internet telnet and ftp access to justify having an account. We have found that many people think they need direct access to the Internet in order to send Internet mail. Since sending Internet mail is possible from any networked machine at Intel, we inform the user how to send mail and this eliminates the need for the account. We do ask that the user have a legitimate business reason for telnet and ftp access before we grant the account.

Second, accounts on Internet accessible machines are set to expire at 6 months. If a user doesn't use the account enough to notice it has expired, it will not be an open door. This is a minor inconvenience to users who need their accounts (especially compared to the benefits).

- Key Lesson #7 – User education is critical
- Key Lesson #8 – Create explicit and enforceable policies

Third, Intel has created a set of Internet Etiquette Guidelines for Internet users (contained in appendix A). The Task Force felt it needed a distinct set of guidelines for a number of reasons: First, policies need to be explicit. Tradition and word-of-mouth fail to carry any legal consequence. Second, existing Acceptable Use Policies[3,4] are too generic. Although most of these provide good general guidelines, they do not deal with circumstances specific to Intel or even specific to a business environment. Third, we've found that Netiquette Guides[5] are good for beginning users, but may not necessarily address behavior problems of the more knowledgeable.

Increasingly, we have found that Intel employees fall into 3 camps: those that know everything about the Internet; those that know about the Internet but feel it's "just like the computer bulletin boards I've used from home"; and those that have heard of it, know that "good stuff is out there," but have no idea how to proceed. Although these groups have very different levels of understanding all indulge in behaviors which need governance.

The experienced user may have had access to the Internet in previous jobs or in college. That previous experience may have been in an environment less demanding than Intel's, since the Corporation emphasizes a stringent work ethic and places heavy demands on employee time. Those employees familiar with bulletin boards may have no clue as to the global community in which they now find themselves, and those new to the 'Net just have no clue. Each needs help understanding the environment.

Experienced users should be informed that Internet use should indeed be work related. Wanting to get to Usenet Newgroups to keep up with discussions on rec.whatever is not an acceptable reason for 'Net access, although needing to stay current with comp.sys.intel certainly is. Experienced users should also understand that their role and responsibility has changed. As students at Wherever.edu no one cared what they said in postings, but people form opinions of a company based on its employee's communications. Disclaimers don't seem to matter, no matter how sincerely stated. Strongly offended readers focus on "intel.com" in mail and article headers.

Half-way knowledgable users need to be educated to the ways of the Internet. These users may be familiar with other forums of computer communication, most likely PC-type bulletin boards, or Prodigy/Compuserve models. These users need to know that their postings span countries and continents, rather than a local community or even the US. They need to learn the jargon and the context of discussion groups. They should "lurk" for a while before jumping into discussions.

Inexperienced users need all the help available. They need to know what kinds of services are available, what the community is, and how to interact with it. With these communities in mind, the guidelines Intel provides fall roughly into those covering technical/security issues, those covering etiquette, and those to help new users. They are broken into categories for electronic mail, mailing lists and newsgroups, ftp, and telnet.

The electronic mail section covers such new user concerns as `SENDING MESSAGES IN CAPITALS`, use of the smiley face :-), and watching punctuation and spelling while not criticizing others' mistakes. Etiquette, such as letting a sender know a message was received (especially when one cannot respond immediately) and having a signature file, is also defined. Issues such as taking care when sending replies, sending plain ascii text (as many Intel users often send PC file attachments in cc:Mail), and being aware of system etiquette on their native system comprise the technical issues addressed. Finally we remind users that electronic mail is unencrypted and easily readable.

The section of the guidelines on Internet mailing lists and Usenet News groups references the section on electronic mail. This is by far the longest section of the guidelines since all employees can send and receive Internet mail. They are also most likely to make mistakes in this area, although in general these mistakes will be less catastrophic than in telnet or ftp. Here, we inform users to disclaim speaking for Intel, and that even if they do, they will represent the company de facto through having "Intel" in the mail header. Along with that technical warning, we direct users to watch verbosity since many Internet sites pay by the byte, to obey copyright law, and to be careful using auto-reply features in mail. We also tell them to change their addresses with mailing lists when they change accounts. There are many guidelines covering straight etiquette: Monitor any group you join for a while, No advertising of Intel products, Don't re-post without permission, Summarize if you survey, Indicate quoted material, No anonymous postings, and No postings about that dying child in England (he got better)! New users are cautioned to make sure the subject of messages is clear in the Subject: line, to think about how much time mailing lists or news groups will absorb, to read the FAQs, to be careful of flaming, and not to go overboard if they're flamed.

The section on ftp leans heavily toward technical issues. The only point of etiquette is that users should type in real Internet addresses for passwords when accessing anonymous ftp sites. The other issues covered: do not deliberately ftp to machines without ftp access, random net-hunting is not approved; observe working or posted hours for ftp sites and observe any restrictions posted at those sites; look locally for ftp materials (where items are posted more than once); and finally don't ftp on the "off chance you'll need the information someday."

The telnet section is even more succinct, covering posted restrictions, using only authorized ports, not not deliberately telnetting into machines with no guest account.

There is a final section, listing a bibliography of Internet resources for beginners. It lists Kehoe[6], Krol[7] , LaQuey[8], and Tennant, et al.[9]. Hopefully, the beginning users armed with the Guidelines, and one of these publications, can survive on the 'Net.

There is another section of the Guidelines listing behavior which is subject to disciplinary action. Here is where our Guidelines differ most dramatically from generic Netiquette guides, since these are areas where we do more than recommend behavior. The guidelines promise action for sending chain letters, for using Intel equipment for personal gain, for sending sexually or racially harassing messages, for unauthorized attempts to break into any system (since Corporate Information Security occasionally gets authorization to attempt break-ins), theft, or copying electronic files without permission, sending Intel confidential materials outside of Intel, and refusing to cooperate with a reasonable security investigation. These guidelines were specifically derived from the Corporate Information Security guideline on mail and from the Human Resources general guidelines. Since this is policy and not procedure, it does not include specific disciplinary actions which might be taken but leaves that for Human Resources to sort out at the time of the incident.

The guidelines were drafted by one person and submitted to an internal mailing list which included the Internet Security Task Force and system managers of machines which have Internet access. This draft gathered comments from "It's fine the way it is" to "Change everything about it". Comments were incorporated into a second draft, which was again circulated to the group. Comments on this draft were minor, although Corporate Information Security made a few specific requests, most having to do with making implicit statements more explicit. (**Mail on the Internet is Not Secure** being the major one.) The final version was sent to the internal mailing list of system managers for distribution to their users. It was also made available for anonymous ftp within the company.

Finally, the policy was adopted as a formal Intel Policy. We did have to get it approved by Intel's legal staff. Now we'd had our policies ratified.

Keeping the Barn Doors Closed

- Key Lesson #9 – Policy transitions can be hard, especially when you have to take something away.

Although we have drawn up new "official" policies, we find that it can be hard to get people to transition to them. It is especially difficult when people lose privileges they once had. For example, we would like to reduce the number of Internet access machines at each site. Getting groups to give up their access is not easy, especially if they have had their own access system for several years. We have found the best time to get people to implement policy changes is after an incident has occurred. While this truly is closing the barn doors after the horses are out, it definitely prevents any more horses from leaving. After implementing the policy on some of the major access nodes, we have had a drop in reported incidents from them.

We need to improve our user education. Although we have created guidelines and even an Intel Internet user guide, it is obvious to us (as indicated by gross violations of Netiquette) that this information has not propagated widely. Getting users to read and understand the policies is a major challenge. One bright spot is a class that Intel has created on Information Security for its employees. Information Security is planning to include the policy in the next edition of its booklet distributed to all employees.

Unfortunately, closing the door to the Internet means keeping some of those resources unavailable to Intel employees. Intel still needs to maintain a competitive edge. In order to allow additional access to Internet resource, we are considering and implementing alternatives. We have implemented an internal ftp machine, which holds internal information for the company, provides mailing list capability, and caches and mirrors external archives. This capability allows us to fill many information needs without having to grant full internet access to the entire company (it also helps us to conserve the bandwidth of our Internet connection). Employees who have one-time needs can send mail to an ftp-admin account with their request and the ftp administrator will search the Internet and mail the results to the employee.

- Key Lesson #10 – Policies exist to serve. They should be changed when circumstances warrant.

Many employees still find our policies limiting. Having someone else search for you is never as satisfying as searching for something yourself. Users have been clamoring to run Gopher, WAIS, World Wide Web clients from their own PCs. We are looking at alternatives like proxy agents for these services. We are also evaluating easing some of our policies for WAIS and Gopher access. The Internet is a constantly changing environment, with new services springing up all the time. We will need to make changes to our policies, but when we do so, we will not ignore the many lessons we learned.

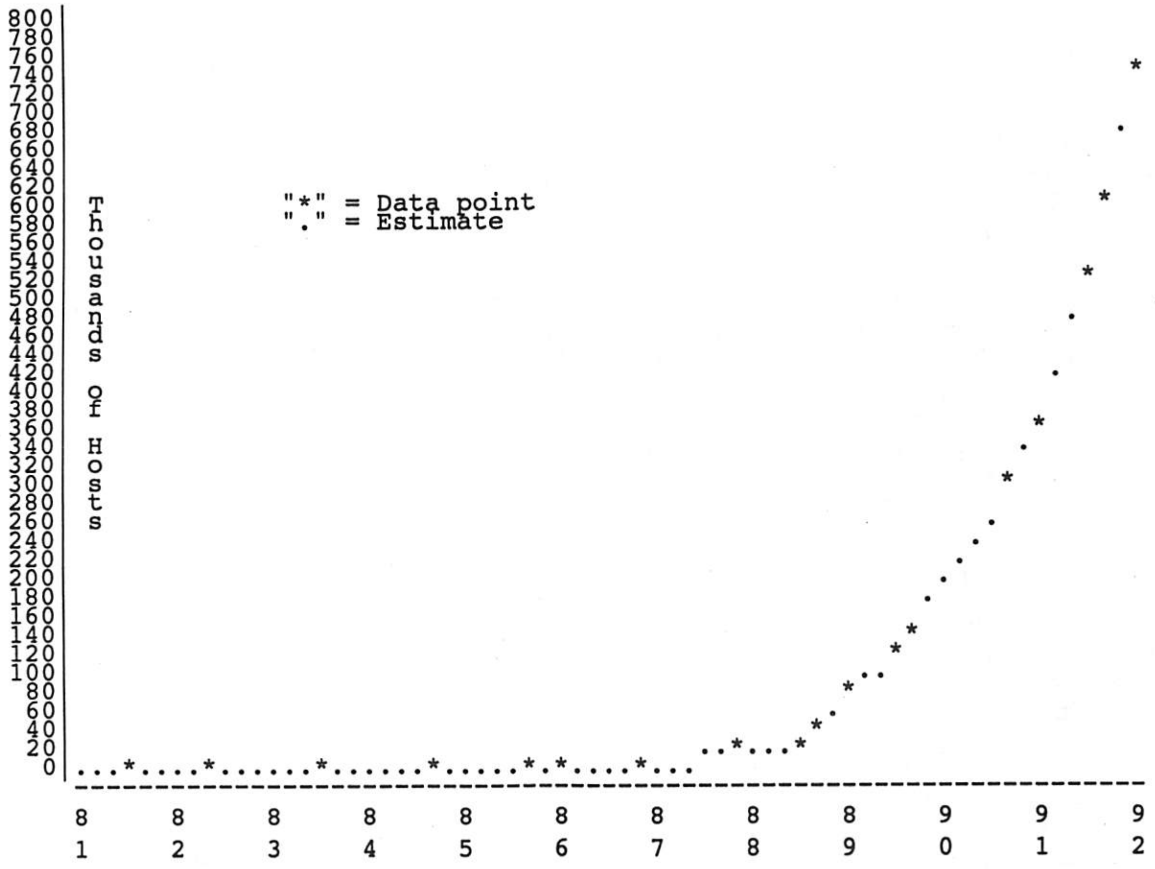

Figure 1: RFC 1296, Internet Growth (1981-1991)

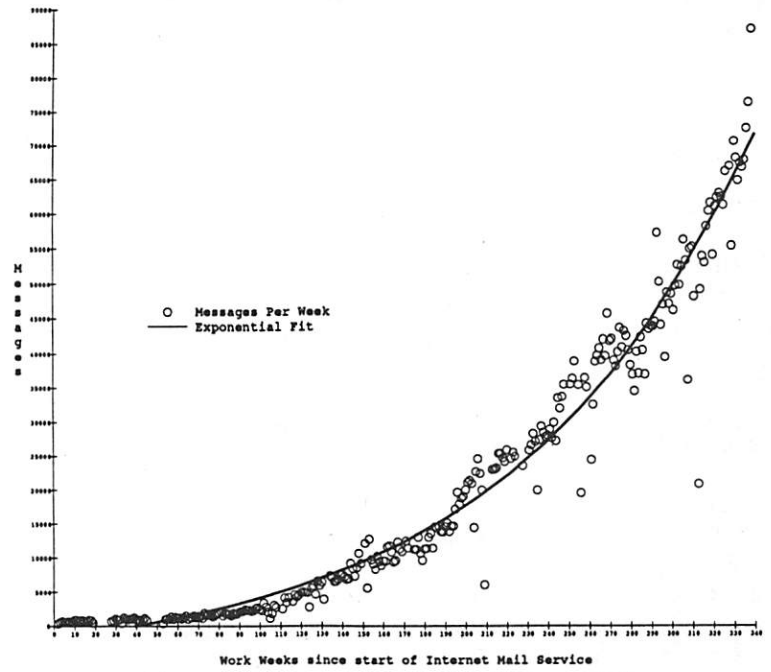

Figure 2: Internet mail by week since 1987

Author Information

Sally Hambridge received her BA in English from UCLA in 1970 and her MLS also from UCLA in 1979. She worked as a contract employee for Xerox. Joining USC/ISI in 1980, she got her first taste of the Internet. She moved to Atari in 1982, then joined Intel in 1984. There, she has been librarian, database analyst, currently runs an internal ftp server. Reach her via U.S. Mail at Intel Corp; SC3-15; 2880 Northwestern Parkway; Santa Clara, CA 95052-8119. Reach her electronically at sallyh@ludwig.intel.com.

Jeff Sedayao received a B.S.E in Computer Science from Princeton University in 1986 and a M.S. in Computer Science from the University of California at Berkeley in 1989. He has worked at Intel Corporation since 1986, spending most of his time running Intel's main Internet gateway. Reach him at Intel Corp; SC9-37; 2250 Mission College Boulevard; Santa Clara, CA 95052-8119. Reach him electronically at sedayao@argus.intel.com.

References

[1] Lotor, Mark. ''Internet Growth (1981-1991); RFC 1296,'' January 1992. Available via anonymous ftp at ftp.nisc.sri.com rfc/rfc1296.txt.

[2] Holbrook, J. P.; Reynolds, J. K. ''Site Security Handbook; RFC 1244,'' July 1991. Available via anonymous ftp at ftp.nisc.sri.com rfc/rfc1244.txt.

[3] ''Acceptable Use Policy for NSFNET Backbone''. February 1992. Available via anonymous ftp at is.internic.net as infosource/nsf-nren-nii-info/nsfnet/acceptable-use-policy.

[4] ''Corporation for Research and Educational Networking (CREN) Acceptable Use Policy''. January 1993. Available via anonymous ftp at cren.net pub/cren-doc/cren.net_use.

[5] Von Rospach, Chuq, Gene Spafford. ''A Primer on How to Work with the Usenet Community''. January, 1991. Available via anonymous ftp at ftp.eff.org pub/internet-info/usenet.etiquette.

[6] Kehoe, Brendan P. *Zen and the Art of the Internet: A Beginner's Guide.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[7] Krol, Ed. *The Whole Internet: User's Guide and Catalog.* Sebastopol, CA: O'Reilly & Associates, 1992.

[8] LaQuey, Tracy. *The Internet Companion: A Beginner's Guide to Global Networking.* Reading, MA: Addison-Wesley, 1993.

[9] Tennant, Ron, John Ober & Anne G. Lipow. *Crossing the Internet Threshold: An Instructional Handbook.* Berkeley, CA: Library Solutions Press: 1993.

Appendix A: The Intel Guidelines

EFFECTIVE DATE OF CURRENT REVISION: 6/93
LATEST REVIEW APPROVE DATE:
NEXT DATE TO BE REVIEWED:
SOURCE FUNCTION: Internet Security Task Force
COORDINATOR: Internet Education
RESPONSIBLE REVIEW MANAGER: Intel Security

1.0 PURPOSE/SCOPE

These guidelines set the standards for appropriate behavior of an Intel employee when accessing the Internet. These guidelines apply to all Intel employees. Intel specifically reserves the right to modify, change or discontinue any portion of the Internet guidelines from time to time at its sole discretion.

2.0 DEFINITIONS

- Cracking – attempting to break into another system on which you have no account, and is treated as malicious intent.
- Netiquette – a word made from combining ''Network Etiquette.'' The practice of good manners in a network environment.
- MIME – Multipurpose Internet Mail Extension. The format for Internet mail which includes objects other than just text.

3.0 GENERAL

4.0 GUIDELINES

4.1 Behavior resulting in disciplinary action.

The following behaviors are examples of actions or activities which can result in disciplinary action. Because all possible actions cannot be contemplated, the list is necessarily incomplete. Thus, disciplinary action may occur after other actions when the circumstances warrant it. Disciplinary actions range from verbal warnings to termination; the severity of the mis-behavior governs the severity of the disciplinary action.

- Unauthorized attempts to break into any computer whether of Intel or another organization. (Cracking).
- Using Intel time and resources for personal gain.
- Sending threatening messages.
- Sending racially and/or sexually harrassing messages.
- Theft, or copying electronic files without permission.
- Sending or posting Intel confidential materials outside of Intel, or posting Intel confidential materials inside Intel to non-authorized personnel.
- Refusing to cooperate with a reasonable security investigation.
- Sending chain letters through electronic mail.

4.2 Behavior considered prudent, good manners, etiquette.

The following behaviors are recommended for sending Internet mail, participating in Internet mailing lists and Usenet groups, ftp, and telnet. Lack of conformance may result in loss of Internet access. These guidelines have been gleaned from a variety of Internet Guides. A bibliography follows these guidelines, and we recommend you acquire one (or more) of these guides.

4.2.1 Electronic Mail (Email)

The following guidelines cover the sending of electronic mail outside of Intel.

- MAIL ON THE INTERNET IS NOT SECURE. Never include in a Email message anything which you want to keep private and confidential. Email is sent unencrypted, and is easily readable.
- Be cognizant of any system etiquette. The computer on which you reside may have quotas on disk space usage. Mail takes up space. It's best not to save every message you receive.
- Do not attempt to send anything but plain ascii text as mail. Recipients may not have the ability to translate Word or WP documents. MIME format messages are encouraged. (MIME=Multipurpose Internet Mail Extension).
- Be careful when sending replies – make sure you're sending to a group when you want to send to a group, and to an individual when you want to send to an individual. It's best to address directly rather than use the reply command.
- Include a signature which contains methods by which others can contact you. (Usually your Email address.)
- Let senders know you've received their mail, even if you can't respond in depth immediately. They'll need to know their mail hasn't gotten lost.
- Watch punctuation and spelling.
- Remember that the recipient is a human being. Since they can't see you, they can't tell when you're joking. Be sure to include visual clues. Convention indicates the use of the smiley face. :-) (Look sideways).
- DO NOT SEND MESSAGES ALL IN CAPITALS. It looks as if you're shouting. Use capitals for emphasis or use some other symbol for emphasis. That IS what I meant. That *is* what I meant.

4.2.2 Internet mailing lists and Usenet News Groups.

All the guidelines covering Email should apply here as well.

- Actively disclaim speaking for Intel. Note that if you use an Intel system to post an article, Intel's name is carried along with what you post in (at least) the headers. The "standard" disclaimers attached to many articles are meaningless if the reader finds the article offensive.
- Remember that some people have to pay for each byte of data they receive. Keep messages to the point without being so terse as to be rude.
- Obey copyright laws.
- Be sure to change your mailing address if your account changes. Do not simply forward your mail from your old account to your new one. This creates a burden on Intel machines.
- Be careful using auto-reply features in mail when you belong to mailing lists. These replies are often sent to the entire list, and most don't care that you're on vacation.
- As a new member of a group, monitor the messages for a while to understand the history and personality of the group. Jumping right into the discussion may make you look foolish if you have no context.
- Do not advertise Intel products. This violates the Internet Acceptable Use Policy.
- Do not re-post any messages without permission.
- Avoid cross-posting whenever possible. When not, apologize, especially if the groups seem to have a lot of overlap. Of course, apologize for any mistakes in posting.
- Do not post personal messages to a group.
- If you survey the group, post a summary.
- Indicate quoted material.
- Do not post any messages anonymously. This is viewed as bad form by the Usenet community and system managers are asked to track down offenders. This wastes Intel's time and resources.
- Do not re-post any requests for a dying child in England to get postcards to get into the Guiness Book of World Records. The child got well, and the category has been removed from Guiness.
- Make sure the subject of your message is clear in the Subject: line.
- Join lists or monitor newsgroups giving thought to how much time these activities absorb. Also for Usenet, look at the news.announce.newusers group. It contains good information on getting started. There are also local Intel groups which are good for new people.
- Be sure to read the FAQs (Frequently Asked Questions) for your group(s).
- If provoked, do not send angry messages (flames) without waiting overnight. If you still think a flame is warranted, label your message with "flame on". If you receive a flame, don't go overboard in reaction.

Remember that not everyone is as polite as you are.

4.2.3 FTP

These guidelines cover file transfer protocol.

- Do not ftp to any machines on which you do not have an account, or which doesn't advertise anonymous ftp services. Random net-hunting is not approved.
- Observe working hours or posted hours for ftp sites. Most sites request you NOT ftp between their local hours of 8-5.
- Don't ftp during your site's prime hours as well.
- Look locally before ftping something from a site geographically remote. Your system manager can help you find the closest site.
- Don't ftp on the off chance you'll ''need it someday.'' Conversely, don't hunt around for ''neat stuff'' to ftp. If you discover that you don't need what you've ftp'ed, delete it. You can always get it again if you discover you do need it.
- Observe any posted restrictions on the ftp server.
- Use your real username and node as your password on anonymous ftp servers.

4.2.4 TELNET

These guidelines cover telnetting to remote systems.

- Do not telnet to machines on which you have no account, or there is no guest account. Do not attempt to telnet deliberately into anonymous ftp servers.
- Observe any posted restrictions on the machine to which you're telnetted.
- Do not try to telnet into miscellaneous ports; use only authorized ports for access.

5.0 Selected Bibliography

LaQuey, Tracy. *The Internet Companion.* Reading, MA: Addison-Wesley, 1993.

Kehoe, Brendan. *Zen and the Art of the Internet.* Englewood Cliffs, NJ: Prentice-Hall, 1992.

Krol, Ed. *The Whole Internet: User's Guide and Catalog.* Sebastopol, CA: O'Reilly & Associates, 1992.

Tennant, Ron, John Ober & Anne G. Lipow. *Crossing the Internet Threshold: An Instrustional Handbook.* Berkeley, CA: Library Solutions Press, 1993.

Our Users Have Root!

Laura de Leon, Mike Rodriquez, & Brent Thompson – Hewlett-Packard Company

ABSTRACT

This paper describes how things work at our site, where users have responsibility for administering their own workstations.

Hewlett-Packard Laboratories is HP's primary advanced R&D laboratory, with about 1000 people and about 1500 computer systems. A central support organization within HP Labs is charged with creating an infrastructure to enable this to work successfully, even though they do not have root access on most systems in the division.

The paper gives some examples of how things are done differently than at other sites, and details what is in place. It also gives some areas where there is work yet to be done.

Introduction

Our users have root! In fact, our users have total responsibility for administering their own workstations. In most cases, we as a support organization do not have user or root accounts on the systems. We attempt to provide the support and infrastructure necessary for this model to work for everyone involved.

This paper is not going to justify why users are responsible for their workstations – it is a fact of life for us, and good or bad, it is not going to change soon.

This situation does have a major impact on what we do and how we do it. This paper will address how we handle things differently than we would if we controlled all (or most) of the environment here. It will identify some issues that we have not handled yet.

Our Site

Hewlett-Packard Laboratories (HP Labs) is HP's primary advanced research laboratory. The Palo Alto site has about 1000 people, with about 900 Unix systems and about 600 PCs scattered through 6 buildings separated by up to two miles. These people are our customers. Most of the people with Unix workstations are the researchers and their support staff.

Most of our computer systems are made by one vendor (HP of course), but they are of several different flavors and types (often incompatible), running different versions of the OS. In addition, there are people doing operating system research running various other systems.

Research Computing Services (RCS) supports as much of this environment as possible. We provide the resources necessary for people to run their own machines.

The Challenge

Our customers want to spend as little time as possible administering their computer systems, they want an advanced working environment with the tools they need to do their jobs, they want to spend as little money as possible, and they want complete flexibility. They do not have to use any particular solution we present, if they perceive it as not meeting their needs or as too much trouble.

We try very hard to satisfy all these, including flexibility. We want maintainable solutions, based as much as possible on a common environment we can convince everyone to use – the environment must be as supportable as possible. We want to help everyone make reasonable use of their resources. We want to enable everyone to do their jobs more efficiently.

Our responsibilities include providing services such as e-mail, news, printing, application support, OS updates, and file sharing, and providing support for the customers when they have difficulty with their workstations, whether as users or as administrators.

Some of the challenges we face, that are different than they would be if we had central control:

- System Integrity – we can't depend that ANYTHING is the way it should be on any system we are asked to help fix.
- Inconsistency – For example, we can't do site hiding because users' names on their workstations may not be the names registered on the mail server.
- User error – having to help users solve problems they have gotten themselves into (the 'rm -r /' syndrome).
- Troubleshooting – we are often called upon to troubleshoot a problem on systems to which we have no access.
- Security – We don't control systems, we don't know how secure they are.

- Not being able to do it ourselves – some things might be easier in batch (like OS updates) but we can't make that decision.
- Sometimes, users don't know what is important – things like running regular backups.

We try to address these problems by providing tools, infrastructure, and skilled support staff. We can make these available, but we can't force customers to install or use them, even if things work better when they do.

Some Things Don't Change

We offer news via NNTP, and people can request an account on a timeshare to read news. This is probably how we'd do news even if we controlled the workstations.

How we handle the network also doesn't change much, at least partially because people usually act responsibly (and systems often break right away if users don't follow network rules). Handling of IP addresses is described a little further on.

Our Organization

Research Computing Services (RCS) has a Customer Support group, that does the front line support. Our customers are free to call on us for any computer related issue; they start by calling or sending mail to the Customer Support Service Desk. The Service Desk dispatches most of the calls to the front line support groups, and the remaining ones directly to the rest of RCS.

Front line support is actually two groups – the System Support Group does basic troubleshooting and On Site Support deals with hardware and other issues most appropriately handled physically at the system in question. The front line support people have a small but growing set of "Standard Answers" available from the rest of the organization – these are descriptions of common problems or situations, and the official way of dealing with them. If there is no Standard Answer for the call, and they can't fairly quickly create an ad-hoc diagnosis, then the call goes to the "technology owner" – a member of another part of RCS.

The other groups in RCS are Network Engineering, Applications, Client Platform, and Site Platform. Every individual in these groups is responsible for some products or areas (the "technology owner"). If no individual has responsibility for a product or gray area, the appropriate manager is responsible. Managers can deal with the question or problem themselves, or assign it to someone in their group. The official level of support for any particular item is allowed to be "none".

Customers can call with any sort of problem and we will help them if possible. This sometimes requires the support person having access to the system. Customers have the choice of giving access or not. It's all up to them. No access pretty much precludes us helping solve the problem. When access is granted, it has ranged everywhere from just reporting by phone the results of what the customer thinks the support person said to type, to giving the support person the root password.

The expectation is that we will troubleshoot, and help the customer implement a solution to their problem. In practice, we often end up fixing things for them, leaving them none the wiser about how things work (at the customer's choice).

We are writing this paper from the point of view of the Site Platform group, which is responsible for the site infrastructure – servers and services that affect the entire site (other than physical networking).

Actually, We Do Have Root Sometimes

Well, yes, we do have root capability on many machines, usually once a day. How this happens is: we have convinced many/most of our users to install a service we provide to "keep important files up-to-date". This service is called "Ninstall Star".

Ninstall is an internally developed client/server network software distribution utility widely used throughout HP. Ninstall(1L) is the client program; it installs software onto the client system according to the installation specifications contained in "packages" on the server. One important feature of ninstall is that it includes a preview option, to see what would be done (installed, updated, deleted, chmod'd, touch'd, etc.) if the package were actually ninstalled. (Ninstall was described in a paper presented at the first LISA conference in 1987, and also a more detailed paper presented at Uniforum, 8-11 February 1988, pp. 41-53.)

What we have done is to encourage HP-UX workstation owners to frequently install every available package from a particular one of our ninstall servers, i.e.:

```
ninstall -v -h hpllan "*"
```

Most users have cron(1) do it nightly.

Some users choose not to run Ninstall Star from cron. These folks usually know what they are doing, and just prefer to occasionally execute this command manually. Some users don't install "*" but instead select for themselves which of the available packages they want kept up-to-date on their systems.

The ninstall preview option allows users to verify for themselves whether any ninstall package is acceptable to install, and as a rule packages should be previewed before installation. Ninstall Star from cron requires users to trust us enough to run it without previewing.

Even though we COULD do anything as root we want on every system that uses this service (but probably only once!), in fact there are many constraints. An implication of being granted root like this for a few moments each day is that we must be very careful how we use this capability. Once the trust given us is lost, it would be very difficult to regain.

Users expect Ninstall Star to not change the behavior of their systems. The purpose of Ninstall Star is to keep up-to-date those files that RCS has determined need to be upgraded independent of OS upgrades. Primarily this includes files that are volatile in nature, or are "infrastructure configuration files". Specific examples include: /etc/hosts (useful for comments contained therein), /etc/resolv.conf, and printer config files. Although we might occasionally update programs via Ninstall Star, this would only happen for "emergency" reasons, not simply because a new program is available.

An example of when it was felt necessary to update binary files was during the infamous Internet worm situation of November 1988. At the time, not all of our workstations were protected by firewall gateways. Our sendmail was secure from this attack, but we were vulnerable to the ftp daemon opening. So, we used Ninstall Star to correct ftpd. Given the seriousness of the situation, we received no complaints about this action.

Another anomalous use of Ninstall Star was for data gathering. We wanted to estimate the amount of disk space in use at HP Labs. So that folks wouldn't perceive us as "snooping", we structured this temporary package to be easily excludable and announced our intentions well in advance, including details of the steps necessary to prevent the disksize script from running or to run it by itself. This also allowed the "non-cron" crowd to participate in the measurement. We then let Ninstall Star run the disk size program for about a week. Again, we had no complaints, and received feedback that people were pleased we had informed them in advance.

We don't handle configuring printers on workstations much differently than we would if we "really had root" – except of course that users choose for themselves whether to keep their list of configured printers up-to-date via Ninstall Star or not.

Currently, Ninstall Star configures all public printers in the buildings nearest to each client system. This means that when a new public printer is deployed, users on systems running Ninstall Star automatically have access to it. As the number of printers is growing, we are evaluating methods to allow users to easily select a subset of all the available printers.

Self-Actualized OS Update

Customers can do it themselves. In fact, they really must do it themselves. Again, this revolves around that major feature of our environment: that our users have root capability on their systems . . . and we do not. Whether and when an OS update occurs is purely up to each individual user. Because it is so much faster, simpler, and less dependent on peripherals any given system may not have (e.g., tape drives), we do all our OS updates over the network. The OS update procedure we provide must be a pull operation, not a push.

Whenever a customer decides to do it – she does it! (In order to maintain a manageable load on the servers, we do limit the number of concurrent update sessions – this limit is high enough it has never been reached yet, but could interfere with a user's preferred schedule.)

One result of users doing OS updates on their own schedules is that we must continue providing each version of the OS almost indefinitely. We basically support only HP computers, which simplifies our lives a bit. Even so, we support three types of hardware running Unix (HP-UX), and currently provide up to four versions of HP-UX for each of these architectures (meaning major OS revisions, the oldest released as long as four years ago).

Also, we always provide HP-UX in four major flavors (combinations of these two option sets): with HP Labs customizations vs. pure vanilla product; and all local-disk-resident vs. approximately 50% offloaded via symlinks to our central server (we call this "nfs-linked").

The algorithm for deciding whether we include any given file in the list to be nfs-linked is based on frequency/importance of use. We nfs-link only "unimportant" files, i.e., files that are not normally used for bootup, not normally used during login, not important for system diagnosis/reconfiguration/fixup, not used during the execution of normal activities, and not even accessed at all most of the time by most users. The file sometimes considered the archetypal member of this set is /usr/bin/banner – a file that is useful when you want it, but infrequently wanted at all and highly unlikely to be mission-critical when you do want it.

The selection algorithm is fairly stringent, which is why we only offload about 50% of HP-UX – for almost any particular system it's easy to find plenty more candidates. Even 50% or so of the (default) OS is quite a bit of space, though – who wouldn't appreciate an instant extra 40 MB of disk space with no hassle and no loss of functionality? And sure enough, most of our users choose to install the "nfs-linked" option. The "local" option exists mainly for those users who remember "like it was yesterday" when the server died for three hours in 1989 :-), and for certain time-share, cluster, or departmental servers.

Even most "local" users agree that local copies of the standard man pages are a waste of space, though, so we never automatically include any man pages in an OS update. [Even so, we do provide packages to get them by update across the network.] We always provide formatted, uncompressed copies of all standard man pages on our two central servers.

Ninstall(1L) is the main tool all our update procedures use. As previously mentioned, this tool is a generalized network software distribution utility that allows greater flexibility than the HP-UX tool update(1m), which is tuned to some particular aspects of HP-UX updates.

We distribute pre-customized default config files with HP-UX updates, so the system boots and works ok after being updated, no matter how different the new OS version may be from the old one. The update process deletes config files that are unchanged from a previously installed version; it saves changed ones with a .OLD suffix and generates a message to check for any customizations that might need to be merged back into the active version.

One feature we have included in our configuration files involves the consolidation of all system-specific information into one place, /etc/SYSTEM.INFO. We modify configuration files and software that need some piece of system-specific information such as TZ or BUILDING to read SYSTEM.INFO rather than having that information hard-coded. Such local information is extracted using the tool developed for that purpose, getinfo(1L), e.g.:

```
/etc/getinfo TZ
```

This single point of administration is convenient for us, but especially important for our root-wielding users who aren't experienced system administrators.

Standardized Systems

One side-effect of our (default) OS update process is that the invoking client ends up standardized, however briefly, to our division's notion of a standard system. Whether the owner subsequently undoes any of these standardizations is beyond our control of course, but mostly they don't (since it's easier just to choose a non-default installation option and not install our customizations at all than to install the default then remove our setup afterwards).

Therefore, this OS update process obviously can be, and often is, used simply to bring new systems up to the standard customization, e.g., newly purchased systems or systems of persons or groups newly brought into our division. We make all our update processes idempotent – there is never any harm in rerunning/restarting any update.

Patches

Along with the HP-UX update procedure, we provide a single-command procedure for installing whatever patches our group (RCS) currently recommends for each version of HP-UX. This patch procedure is described in documentation mailed to each client during the update process.

Patches always come unexpectedly, with no warning or schedule. The idea behind this recommended-patches command is that it should be run by cron every week to keep the system up to date with any recommended patches that may have come along in the meantime. Practically speaking, it's pretty disruptive to have the system patched and rebooted without warning by cron (if a recommended patch involving new kernel libraries comes along some week).

As with most of our update procedures, we provide a preview option which tells what would be done if the update were actually performed. We recommend that users should run the preview of recommended-patches every week via cron. If something turns up some week, they can just install the current recommended patches at their convenience.

As with everything we provide, whether any user installs the recommended-patches or not is up to them. To make it easy for the inevitable users who want to take advantage of the functionality and simplicity of the recommended-patch procedure, but want a different list of patches installed, this process is split into two parts, like most pieces of our OS update process. First the tools and lists are installed, then the tools are run against the lists. This way any user may install the tools first and modify or replace the default lists, then proceed with the second step that will act on his own lists. But these steps are only for users who desire this special feature; by default, the whole process just happens in one continuous step after the user specifies the single initial command.

We provide an ninstall package for every available patch, so users can easily install any patch they wish; the recommended-patches package is just a wrapper to make installation of the most important set of patches so easy even our most naive users will be able to do it themselves.

Applications

We also provide various applications for installation from our servers, such as Emacs, Epoch, FrameMaker, Lotus 1-2-3, LaTeX, LAN Manager, etc. They are all available using the same interface as HP-UX updates; like OS updates, "nfs-linked" and "local" versions are available for all. As with OS updates, the nfs-linked version is also most popular for applications. For example, consider this extreme case: FrameBuilder "local" installation = 85 MB, "nfs-linked" installation = 5 KB.

During OS updates, we try to determine if newer versions of any applications exist and, if so, attempt to update them automatically. Again, as with all our services, users can pull a new application onto their systems whenever they wish.

Applications (and other versions of HP-UX itself) are available from sources other than the RCS servers, so a flip side of this coin exists, too. Users control their own systems and they do install applications from those other sources. Sometimes this causes files to get overwritten, even important configuration files that we have worked hard to set up just right. Sometimes this causes systems to break in new and unexpected ways! So another result of only system owners controlling their systems is that troubleshooting is more complicated for us – we can never take anything for granted about the system setup. We always have to check even the most obvious, standard things: we've had printers break because spool directories weren't owned by 'lp'.

Documentation

We always have to write documentation for users who are not "real" system administrators.

One of the hurdles we have that is no doubt different from most other places is that our update procedure must result in a working system regardless of whether the user reads the documentation or not.

A fact of life for us is that usually a large percentage of users will not read it, and unfortunately not because they already know what it deals with – our users are not the sort who attend LISA! One of the Laws of Nature we have discovered (as have many other groups like ours) is that the number of users who will read any piece of documentation is inversely proportional to the length of that document. And, for any operation as complex as a complete OS update, it takes an awful lot of documentation to impart all the information that might be important to different users – all the more so since our users are so diverse.

So, we try our best to ensure users actually read the documentation. One way is to pare documents down to only what is needed for that particular phase. A rarely attained, though always remembered, goal is to have nothing longer than one (60-line) page.

File Sharing

We have no global home directory sharing (or even semi-global). All system owners feel (and are!) free to arbitrarily add and delete users at their own whim, so we have always considered it out of the question to maintain a common UID space. We do keep our central systems synchronized, but only one group is using these as their master list. They send us mail with a list of information, and we assign them a UID. This service is available to everyone, but it is not commonly known, and is fairly awkward to use.

The lack of a global UID space certainly complicates providing a service like home directory sharing. So far there has been little interest in this since all users have their own offices, their own workstations in those offices, and no perceived need to share home directories.

Different work groups do occasionally experiment with these notions of file sharing or common uids, but so far none of these experiments has caught on and spread.

We do have a form of global file sharing. We have public disks with minimal security that everyone can mount via NFS and LAN Manager for read/write access. These are available to anyone in the division with a workstation or a networked PC.

This is useful for making some files available to a few people, but the lack of security makes it less useful than it might be. In addition, ownership information is meaningless because of the lack of global UID space – UID 1234 might map to user 'A' on one system, and user 'B' on another.

A shared disk works as an alternative to mailing a file to a large group of users, for transferring files from PC to workstation or back, for putting out files people may or may not be interested in, and so forth. For example, someone might send a message that says "file ABC describes the plan for XYZ.. If you are interested, read it and send me your comments", rather than sending file ABC to the full list of people.

Such shared disks are not useful for permanent storage, moving your home environment around, etc. Old files are subject to removal from these disks.

Many groups also have their own departmental servers, but we generally have no hand in either setting up or administering these, i.e., they are just among the undifferentiated set of client systems on which we have no access.

DFS (OSF's Distributed File System) with Kerberos authentication might help us by providing workstations that don't currently trust each other a way to do a truer level of file sharing, with some degree of trust in the security of what is being shared. Kerberos authentication seems like it may make many services easier to provide.

To distribute software, share system files, and support the "nfs-links" previously described, we have duplicate read-only file servers that have complete copies of all HP-UX and application bits we support. Essentially all workstations mount these disks (the particular ones relevant to each client system's architecture and OS version, that is); these are the disks to which we "nfs-link" various system files to save local disk space.

E-Mail

E-mail is another area where we have been successful in putting together a plan that works.

We have two completely different e-mail systems in use – much of the non-technical staff uses HPDesk, a proprietary e-mail system that runs on HP's MPE systems. An unrelated group of people at a central IT (as "MIS" is now known) facility manages HPDesk and the gateway to and from it.

Of the people who get their mail on Unix, most have mail delivered to their workstations. Timeshare facilities are available if customers want to get their mail in a central location; these are most often taken advantage of by users with PCs on their desks. Sometimes customers use the timeshared mail facility so they can have their mail in a central location where it is backed up, and they don't have to worry about keeping up with the software (and the patches).

We maintain a central list of mail aliases for everyone in the division. firstname_lastname is valid for everyone. We also set up (register) a username alias for everyone that gets mail on Unix, has an account on a Unix timeshare system, or requests a username alias.

Traditionally at HP Labs, the registered mail alias has been the user's last name (with a first initial prepended if needed for uniqueness). We recently reevaluated this policy and found there was no good reason for it, and that the policy resulted in extra complexity – quite a few people who wanted a username other than their last name were using their username of choice on their workstation, and setting up a local alias for the registered alias. This complexity works just fine, but it is nice from a global perspective to have the name a person sends from match her registered (user@hpl.hp.com) address.

We also maintain a number of mailing lists. We have a set of nested departmental mailing lists, so that anyone needing to send mail to a department or organization can send mail to the lists we maintain. The top level lists are moderated to avoid junk mail to everyone.

We periodically check these lists against departmental organization charts. Some departments provide us charts more often than others. Those organizations have more up-to-date lists. When we check our mailing lists, we find out if there is anyone we don't already have a firstname_lastname alias for. In practice, it seems that people get themselves signed up fairly promptly.

Our standard workstation sendmail.cf treats any mail for names not on the system (in either password or aliases file) as user@hpl.hp.com, and sends it to the mail hubs to be resolved. This sendmail.cf is among the files kept up-to-date by Ninstall Star.

The mail hubs accept mail for hpl.hp.com, so users inside or outside don't need to know what system someone gets their mail on to send to them, as long as they know the registered username or firstname_lastname for that user.

Workstations at HP Labs are not accessible directly from the Internet outside HP. We arrange for mail addressed to user@system.hpl.hp.com to get to that system via Mail Exchanger (MX) records that point to a set of gateway systems.

We also use MX records to direct mail for down systems to a mail server for queuing. From there, we can manually redirect it if necessary. This means that users don't have to worry that mail will bounce if they turn their workstations off for a weekend or their system crashes. If a workstation will be down for a while, we can change aliases to point to a new system (a coworker's system, say, or a timeshare system). This won't catch mail sent directly to user@host, but we have scripts we can use on the server on which mail is queuing to redirect it to the new host (or new user@host).

The big picture of MX records for each host looks like the list in Figure 1. MX records are tried in order of preference, beginning with the lowest; we set the first one to deliver mail to the system itself. If this fails, due to the sending host being on the other side of a firewall or the system being down, the next MX will be tried.

The second MX record points to our internal mail hub, where mail will be queued if the destination system is down. If the sending system isn't an internal HP system, connecting to this mail hub also will fail. If this happens, or the mail hub is down, the next MX record will be tried.

The next record points to our main external mail gateway. It will accept the mail and deliver it internally. The only reason to try the next MX record should be if this mail gateway is down.

```
hplmango.hpl.hp.com    preference = 10, mail exchanger = hplmango.hpl.hp.com
hplmango.hpl.hp.com    preference = 20, mail exchanger = hplms2.hpl.hp.com
hplmango.hpl.hp.com    preference = 25, mail exchanger = hplms26.hpl.hp.com
hplmango.hpl.hp.com    preference = 30, mail exchanger = hplabs.hpl.hp.com
hplmango.hpl.hp.com    preference = 35, mail exchanger = hplb.hpl.hp.com
```

Figure 1: MX Records for a system (hplmango)

The next MX record points to a backup gateway in another building.

The last record points to a backup gateway in another country.

Although we suggest each user advertise his address as name@hpl.hp.com, and we accept mail for this form of addressing, we can't implement site hiding. (Whether we would want to, if we could, is another story.) As pointed out before, users may register one alias with us and have a local alias pointing to their real username (either because in the past we wouldn't give them the alias they wanted, or because the alias they wanted already belongs to someone else). Customers can (and do) set up mailing lists and other accounts on their workstations, so, return addresses would not work if we took out the hostname portion of the address.

Networking

An area that occasionally has problems due to the lack of central control is that of adding a new workstation to the network. Under a central system, we would know when a new system was to be connected to the net, and could manage that process proactively for both hostname and IP address assignment.

As it is, when a customer gets a new workstation, she is responsible for requesting a new hostname and IP address. The main problem is when a user hasn't planned for the new arrival and wants to get "on the air" immediately. The usual turnaround on new IP addresses is twenty-four hours – even if an address is assigned in minutes, the nameservers won't all know the new address immediately.

Sometimes there are problems when a workstation is "cloned" and an existing hostname is duplicated, because this leads to a duplicate IP address being used. This does not happen very often, though, and is usually easy to detect. It is also fairly easy to remedy. Also, since we still have a lot of ThinLAN, users can add their own network drops. But this also doesn't happen often and any problems that do occur are easy to find and fix.

Security

Security presents something of an issue. We have no ability to force any level of security on the workstations – but then, forced security isn't the best kind anyway.

So, what do we do?

We control the perimeter – We have a firewall between HP's internet and the Internet, and we have centrally administered dial-back modems. Phones are centrally controlled by an unrelated department – you can't get a phone line in without going through the appropriate channels and local modems on workstations are forbidden without appropriate paperwork. We haven't caught anyone with their own unauthorized modem yet.

Most people get their OS from us, so we can fix bugs there before they install it. We also could use Ninstall Star if the problem were big enough.

We try to educate people – usually all at once, around the time of an internal audit. We will be trying to do a better job at this, using channels like our newsletter.

We make tools available. At this point, not many, but there are plenty easily available in HP. We just need to make pointers to them available.

The responsibility for security lies with the owners of the individual systems and their management – just like it does for the security of the papers in their filing cabinets or on their desks. We are available as consultants, and we try to provide as secure a site-wide environment as possible.

Backups?

Users are responsible for their own backups, but some users don't realize the value, or necessity, of them. And even for those who do, there is often something just a little more important to do right now . . .

Most work is done on local disks on standalone workstations. A VERY rough guess is there is about 1 TB of disk space on the Unix systems, of which about half (500 GB) is data.

We provide a script with the OS to do backups to a local tape drive, if the system has one. Many do, some do not. This script can do full or incremental backups of user files. It has minimal local customization that needs to be done (name and type of tape drive, what needs to be backed up). This is no longer good enough.

We are working on better solutions.

We plan to have some sort of subscription network backup service. Customers will have to push data from their workstations, since they probably won't allow us the unrestricted access that complete backups require. We also would like to have a restoration mechanism that will allow customers to easily restore their own files, yet will be secure enough to keep them from getting at anyone else's files.

Users' Alternatives

Of course, there are some people who don't want to manage their own systems – either because of the time it takes, or because it is a subject they are not interested in dealing with.

We do have some alternatives for those customers. They aren't perfect, but they fill some of the bigger gaps.

We have timeshare systems where customers who just want to read and send e-mail and news, do some basic text processing, and so on, can spend most of their time.

Of course, they still need something on their desks – in the far past, this would have been a terminal; more recently it is likely to be a PC or HP workstation.

The workstation might just have a basic OS, with the customer logging on to a central system to do most of his work. He still has to manage the workstation, but there isn't much of a problem managing a workstation that isn't customized and has neither data nor applications on it.

Many of these customers prefer a PC – the applications they want are there, and timeshare access enables them to get any Unix services they want. Of course, there are issues with PC management, but that is beyond the scope of this paper.

If a workstation is going to be used only for windows to log into a shared system, why have a full workstation at all? This brings us to our most recent plans – X terminals. A customer can arrange with us to boot her X terminal off a shared system, so that all she has to do is set it up on her desk. She can run her window manager on the server or locally. We are just beginning to work with this X terminal plan, so we aren't sure what the final form will be.

An option of allowing customers to buy an increased level of support has been discussed before, but died for lack of funding. We may try this again eventually.

However, all of these alternatives apply to a relatively small set of our users. Most of our Unix users demand the flexibility and power of managing their own workstations, and find the associated admin work a small price to pay.

Conclusion

Our set up really does seem to work for us.

There are a couple of reasons for this. One is that we have an infrastructure in place. We've had to work at this, but then we also would have to work at putting an infrastructure in place if we did have control – the constraints would just be different.

Another factor is that most of the users here are highly educated engineers and scientists who are sophisticated computer users and capable of managing their own systems.

Probably the most important reason is that our customers accept their responsibility. For example, there is no question of them blaming us when they lose a file because they didn't do backups, since they don't see us as responsible. They usually do make intelligent choices, and we are here to help them do that, to give suggestions and to make things possible.

Author Information

Laura de Leon, Mike Rodriquez, and Brent Thompson are all members of the RCS Site Platform group at the Palo Alto site of Hewlett-Packard Laboratories. They all have the same paper mail address, which is Hewlett-Packard Company, P.O. Box 10490, Palo Alto, CA 94303-0969.

Laura de Leon is a Systems Administrator and Technology Specialist. She has worked at HP Labs for 3 years, since she graduated from Harvey Mudd College, where she received a BS in Mathematics with a Computer Science option, and picked up system administration experience on the side. She was member of the SAGE interim board. She can be reached at deleon@hpl.hp.com.

Mike Rodriquez is a Senior System Administrator by title, and the primary architect of the HP Labs site printing, DNS, and Usenet configurations in reality. He has been programming and administering various flavors of Unix since 1984. Hobbies include traveling to attend concerts and playing with his Multi-Media PC at home. His e-mail address is rodriquez@hpl.hp.com.

Brent Thompson is also a system administrator and software developer. During the past seven years he has primarily been focused on Unix OS updates and system file sharing, and is the primary architect of most aspects of these at HP Labs. His major hobby is growing rare fruits and chilies. His e-mail address is thompson@hpl.hp.com.

LUDE: A Distributed Software Library

Michel Dagenais – Ecole Polytechnique de Montréal
Stéphane Boucher – Bell-Northern Research
Robert Gérin-Lajoie – Universite de Montréal
Pierre Laplante – Centre de Recherche Informatique de Montréal
Pierre Mailhot – Universite de Montréal

ABSTRACT

Numerous software packages are being used and updated regularly on most computer systems. Installing all these software packages is a formidable task because each one has a different procedure for compiling or for placing the files required at run time. The LUDE (Logithèque Universitaire Distribuée et Extensible) software library is an organization for installing software packages, a set of tools to install and uninstall software packages and browse their documentation, and a number of FTP servers offering over 100 pre-installed freely redistributable software packages. It offers functionality and flexibility not available in existing systems.

Introduction

The LUDE software library is a joint project of the Computer Science and Operational Research Department of Université de Montréal (iro.umontreal.ca), of the Electrical and Computer Engineering Department of Ecole Polytechnique de Montréal (vlsi.polymtl.ca, info.polymtl.ca...) and of the Centre de Recherche Informatique de Montréal (crim.ca). The LUDE project was initiated in December 1991 to address the following goals:

- Serve heterogeneous systems.
- Let each disk server decide, on a package per package basis, if it wants a network access or a local copy of the executables and/or source code.
- Provide access to new software packages without editing user configuration files (.login, .cshrc).
- let more than one version of a given package coexist during transitions.
- Keep each software package in a separate subtree to ease the management of disk space and prevent name conflicts.
- Make all the documentation easily accessible through a single user interface.
- Let several independent organizations cooperate by sharing software package installation.

In this paper, an overview of the capabilities offered by Lude is presented. The detailed reference manual is found in the GNU info files that accompany the Lude distribution. The next section discusses where and why this project started. The following section reviews existing systems that address the problem of software distribution. Then, a section presents typical client server organizations. Next, the basic file tree organization is presented and Lude installation tools are described. The paper ends discussing status and availability of Lude as well as possible future developments.

Motivation and Organizational Context

The iro.umontreal.ca domain serves 45 professors and 600 students. It contains about 90 UNIX workstations from Sun, DEC and Silicon Graphics as well as 70 MacIntosh and 15 IBM PC compatible network clients. There are 600 user accounts and 8 full time hardware and software support staff in addition to one local administrator for each of the 8 computer laboratories.

A number of subdomains of polymtl.ca are using Lude and serve approximately 48 professors and 500 students (Electrical and Computer Engineering graduate and undergraduate students). There are approximately 100 UNIX workstations from Sun and HP/Apollo, 6 MacIntosh and 20 IBM PC compatible network clients. There are 700 user accounts, 5 full time technicians that work mainly on UNIX support, two research engineers with significant activities in software support and one departmental network analyst.

The crim.ca domain serves 400 users, researchers, software engineers, support staff and external users. It contains approximately 100 UNIX workstations from Sun, DEC, IBM, HP, NeXT and Silicon Graphics. The CRIM focuses its research activities along the following areas: Knowledge-based systems, Speech understanding and signal interpretation, Software engineering, Parallel architectures, Computerized control of industrial processes and computer vision, Teleinformatics and computer networking, Computer-assisted training environments and user interfaces.

All these sites had in common a large number of workstations used to support scientific and teaching activities. In particular, many software packages had to be compiled and installed independently at each site. Monthly meetings were held at iro.umontreal.ca to discuss software packages, networking and system administration. Representatives from polymtl.ca and crim.ca were invited to these meetings and became aware of the common problems and proposed solutions.

Following informal discussions, Robert Gérin-Lajoie, Pierre Laplante, Stéphane Boucher and Michel Dagenais decided to meet and attack the problem of organizing and sharing /usr/local. When this paper was submitted to LISA, an anonymous reviewer suggested a discussion of the problems involved in joint projects. This was probably left out unconsciously from the first draft, not because it is not an interesting question, but because of the difficulty of jointly writing such a section.

A few points are certainly noteworthy. During the 18 months when Lude was conceived and developed, 2 of the original 4 team members changed jobs. Each site went at least once in a state where because of internal changes (new servers, moving altogether) or personnel leaving they could not put any resource on the project for several months. In such a joint project, there is no hierarchical links between the members and one cannot impact much on the priority level assigned to the project by other team members.

Because of the combined experience of the team members, the final Lude organization is more mature and perhaps simpler than what any single member could have achieved alone. Moreover, the approval of members with different backgrounds brings confidence in the soundness of the proposed organization. On the other hand, in many cases the tools had to support everyone's favorite option (internationalization, minimization of symbolic links...). This made the tools more powerful, perhaps more complex, but certainly more difficult to implement.

During the design and implementation, discussions were held by phone, private e-mail and mailing list and several times minor communication misunderstandings arose. For example, one would understate his reluctance to a feature or his difficulty to meet a deadline with some team members more than others. The result would then be a skewed vision of a problem and of the best remedy, between the different team members.

Several other interesting joint development projects could have been pursued: user accounts management, backups... However, sharing software packages was probably in retrospect the best choice. Indeed, each site becomes immediately aware of the interesting packages used or developed by the other sites, including system administration tools.

Existing Systems

A number of systems were developed for managing the installation of software packages such as Xhier[1], Depot[2, 3], AUTOLOAD[4], and lfu[5] but none was found that met our requirements in terms of flexibility, heterogeneous support and, in particular, documentation indexing and browsing capabilities.

One such system, Xhier[1], was developed at the University of Waterloo. It has been used on a large scale and automates even the editing of system files like /etc/inetd.conf, adding required dummy user accounts. On the other hand, it was still evolving, used server initiated transfers and is relatively complex[1]. Furthermore, it cannot be redistributed because of license restrictions.

Lfu[5] was developed at University of Edinburgh and offers a relatively simple but somewhat inflexible organization. Indeed, the basic organization is a tree of servers with one master server for each architecture. In addition, it cannot accommodate easily more than one version of a software package.

Depot[2, 3] is the only system known to us that has been installed at more than one site. It evolved significantly from the first version and is now relatively close to the Lude organization. Indeed, each software package is in its own directory and symbolic links are created in /usr/local/bin, lib, include... towards the files exported by each package. Depot introduces the notion of package collection, not used in Lude, but does not manage multiple architectures. Furthermore, more than one version of a software package cannot easily coexist on a single computer.

Servers and Clients

The LUDE software library enables a large number of sites to pool the software packages compiled by their system administrators. Each computer can act as a client and/or a server as it desires. A client only needs a network connection to a LUDE server (such as the Internet). A server needs to install software packages as described in the LUDE documentation and to export them through NFS (Network File System) or FTP (File Transfer Protocol). A client may represent a more or less heavy load for a server:

- A client takes a complete copy whenever a new package is available and remains autonomous thereafter. This represents a light load and can be performed between distant sites.
- A client takes a copy of the install and run subtrees and maintains a symbolic link to the source code on the server. If access to the source code is relatively infrequent, this is not a very heavy load either.
- A client only keeps symbolic links to the server for the source code as well as the run

time for most software packages. Thus, each time one such package is accessed, the server is involved. This is only acceptable if the client and server are very close, on the same network and in the same organization.

Typically, a multi-level client server organization will be found:

- Public servers allow clients from around the world to take copies of their packages through FTP or NFS. Some usage restrictions may apply if the load is too high on these servers.
- Departmental servers regularly interact with public servers to keep a large set of up to date packages. Department clients may then use these packages either by taking a copy or even through symbolic links for the infrequently used packages. In some cases, a departmental server can also be a public server.
- A local server takes a copy of frequently used packages from the departmental server and perhaps keeps symbolic links for other packages. The source code for these packages is normally accessed through a symbolic link to the departmental server or to a nearby consenting public server.
- Individual workstations may simply mount /usr/local from the local laboratory server.
- Notebook computers copy packages according to their upcoming needs for standalone, nomadic, operation.

The compiled binaries for a software package will differ according to the target architecture and operating system. The two together form the class of the target system. The following classes have been registered up to now:

dec1.2_alpha	sony4.0_68030
hp8.0_s200	sun3.5_68010
hp8.0_s800	sun3.5_68020
ibm3.1_rs6000	sun4.1_sparc
linux0.99.10_386	ultrix2.1_uvax
pyr5.1_mips	ultrix4.1_mips
sgi4.0_mips	vax4.3_vax
sol2.1_sparc	

It is important that the same names be used throughout the various LUDE software libraries on connected servers; currently, the mailing list lude@iro.umontreal.ca is used for this coordination.

File Tree Organization

In /usr/local, the usual directories are found, in addition to server and soft, and have the following content:

- bin: symbolic links to executable files (adequate operating system and architecture class, default version). For example, emacs points to the file /usr/local/soft/emacs-18.59/run/poly-/sun4.1_sparc/bin/emacs.
- lib: symbolic links to files used at execution time by the package. It can be libraries of compiled modules, fonts, macros or even executable files called within a package. As an example, m3 points to /usr/local/soft-/modula3-2.11/run/poly/sun4.1_sparc/lib/m3.
- include: symbolic links to include files like declarations of procedures and data structures for library modules. For instance, m3 points to /usr/local/soft/modula3-2.11/run/poly-/sun4.1_sparc/include/m3.
- man: symbolic links to man pages. For example, man1/emacs.1 points to /usr/local/soft-/emacs-18.59/run/poly/sun4.1_sparc/man-/man1/emacs.1
- info: symbolic links to hypertext files as used in the GNU project.
- doc: symbolic links to unstructured documentation and to software description files in Internet Anonymous FTP Archive format (IAFA-PACKAGES). For example, lude-1.6/IAFA-PACKAGES points to /usr/local-/soft/lude-1.6/install/IAFA-PACKAGES while lude-1.6/README points to /usr/local/soft-/lude-1.6/run/crim/sun4.1_sparc/doc/lude-1.6/README.
- server: symbolic links or mount points to accessible lude servers. For example, poly could be mounted on the directory lude.polymtl.ca:/usr/local/soft.
- soft: one subdirectory for each locally available software package. For example, one finds there emacs-18.59, modula3-2.11...

Thus, each software package is placed in its own subtree in /usr/local/soft. Moreover, every version of a software package is treated as a different package with its own subtree. The unique name of a package is then formed by the concatenation of its name and version number (e.g., emacs-18.59, modula3-2.11). This enables more than one version of the same software to coexist peacefully during transitions and simplifies the management of disk space since each package is kept separate.

However, several modifications (or minor versions) may exist for a package; these usually represent minor modifications to the original source code (for instance in the Makefile). Most often, a single modification is needed and is named after the person or the site performing the compilation.

Inside the subtree, three subdirectories are present src (original source code and modifications), run (everything needed at run time, possibly for several platforms and modifications) and install (description of the package and possibly special actions related to installing this package), as well as a file, history, which traces where this package was copied from. When a software package is installed, symbolic links are created in /usr/local/bin, lib... and point to files or sub-directories in the run subtree of the package.

In more details, a software package subtree contains, for example for a modification named crim and the class sun4.1_sparc, the following subdirectories and files:

- history: actions performed to copy/link locally this software package, for tracing purposes.
- src/orig/*: all the files exactly as they were in the original source code distribution.
- src/crim/*: all the files added to or changed from the original distribution in order to create the crim modification. Often this directory simply contains a modified makefile.
- install/IAFA-PACKAGES: software package description in IAFA format.
- install/crim/LUDE: description specific to the crim modification.
- install/crim/sun4.1_sparc/LUDE: information about who compiled the sun4.1_sparc class, and when, for the crim modification of this software package. Additional files in the same directory may specify files that do not require symbolic links in /usr/local/bin, lib... or actions to perform before and after the local installation.
- run/share: files common to all modifications.
- run/crim/share: files common to all classes within the crim modification.
- run/crim/sun4.1_sparc/bin, lib, include, man, info, doc: all the files required at run time for the sun4.1_sparc class of the crim modification. These files will have symbolic links in /usr/local/bin, lib... pointing towards them. Often the man, info and doc subdirectories in run/crim/sun4.1_sparc will be symbolic links to the corresponding directories in run/crim/share to allow transparent sharing of architecture independent files.

Installation Tools

Lude is both an organization, described in the previous section, and a set of tools. The lude command is a tool that can copy a software package from a server and install symbolic links in /usr/local/bin, lib...; it can also unlink and remove a software package. For example, to install modula3-2.11, modification poly, from the lude-poly server, linking the source code and copying the run time, using the default class for the local machine, the following command is used:

```
% lude -copy run \
    -link -software modula3-2.11 \
    -modif poly -server lude-poly
```

The ludeadm tool is used to create an empty package subtree, separate the local modifications from the original source code and release the compiled package when it is ready for public consumption. The sequence of commands shown in Figure 1 is typically used to compile a new package for Lude.

The ludeindex tool organizes and indexes the documentation. It can generate a keyword database

```
% ludeadm -create -software emacs-19.4 -modification crim
% cd /usr/local/soft/emacs-19.4/src/orig
% ftp prep.ai.mit.edu
ftp> cd pub/gnu
ftp> binary
ftp> get emacs-19.4.tar.Z
ftp> quit
% zcat emacs-19.4.tar.Z | tar xf -
% cd ..
% mv orig/emacs-19.4 .; rmdir orig; mv emacs-19.4 orig
% ludeadm -software emacs-19.4 -modification crim -duplicate
% cd crim
% vi makefile
% make all install
% make clean
% cd ../..
% ludeadm -software emacs-19.4 -modification crim -unduplicate
% vi install/IAFA-PACKAGES
% vi install/crim/LUDE
% vi install/crim/sun4.1_sparc/LUDE.lock
% ludeadm -software emacs-19.4 -modification crim -release
% mail -s emacs-19 lude@iro.umontreal.ca
```

Figure 1: Compiling a new Lude package

for man pages (using the catman command) as well as create a Wide Area Information System (WAIS) database using the synopsis and the first description paragraphs of each man page. It also creates a main menu for GNU info files. Finally, it creates a World Wide Web (WWW) html file for each software package, from the corresponding IAFA-PACKAGES file. The html file also contains hypertext links to the man pages, info and doc files that come with the package. Furthermore, man pages and info files are converted on the fly to html format upon access; the conversion preserves the info hypertext structure and handles adequately the SEE ALSO section of man pages.

Ludeindex can index not only locally installed software packages but also those on remote Lude servers, indicating their availability in the html file. This way, software packages can easily be found either through a main menu or through keyword searches. Moreover, the associated documentation is readily available through hypertext links using the same browsing tool. Even more, chances are that many users are already familiar with WWW browsing tools since they are increasingly used to access public databases such as university course and staff directories.

Status and Availability

The LUDE tools are written in Perl and amount to 5000 lines of commented code. The user manual is a 2000 lines info file. All text messages are kept in a separate file to offer multi-lingual support. At current time, both English and French are fully supported. The LUDE tools and associated documentation are freely redistributable under the terms of the GNU General Public License. They can be obtained through FTP from ftp.crim.ca:lude-crim/lude-1.6.

There are at least three public LUDE servers in operation offering more than 100 different software packages: ftp.crim.ca, ftp.iro.umontreal.ca, ftp.vlsi.polymtl.ca. Three mailing lists are used to coordinate the activities surrounding Lude: lude-request@iro.umontreal.ca to subscribe/unsubscribe, lude@iro.umontreal.ca where discussions and announcements take place and lude-bugs-@iro.umontreal.ca where bug reports should be sent. Nine countries are currently represented on the lude@iro.umontreal.ca mailing list.

Over 1600 software packages (source and/or executables) were downloaded from ftp.iro.umontreal.ca and ftp.crim.ca from the Lude tree. This covers the period from the 23rd of June to the 25th of August 1993.

Slightly over 1000 different usernames (representing perhaps 900 different users) did perform these transfers. All of the following top-level domains were represented: at, au, be, br, ca, ch, cl, com, cs, de, dk, ec, edu, es, fr, gov, gr, hk, hu, ie, il, in, it, jp, kr, mil, mx, net, nl, no, nz, org, pt, se, sk, th, tr, tw, uk, us, ve, za.

Surprisingly, a good proportion of the transfers concerned only source code. It seems to indicate that a major use of Lude is to serve as an extensive source code library, including modifications required to install each package on some architectures. Indeed, even though a site may not want to copy executables, it may copy the local modifications performed to compile the package on the target architecture, verify that these modifications are sensible and perform the compilation; this still represents considerable savings as compared to starting from scratch, while not compromising security.

A publically accessible WWW server demonstrates the documentation indexing and organization achieved through the Ludeindex tool: the "lude list" and "lude index" menu items in http://froh.vlsi.polymtl.ca:80/usr/local/lib-/WWW/default.html. A publically accessible Gopher server, gopher.crim.ca item RISQ/Lude, provides access to the lude and lude-bugs mailing lists and maintains a list of Lude servers.

Conclusion

The Lude project significantly reduced the time spent compiling software packages. Thus our sites were able to offer a much wider selection of up to date software packages. Managing the local installation of packages through lude is much simpler and the documentation is now well organized. All these immediate benefits demonstrate the success of this project. A secondary benefit is that each site is now more aware of good things happening at the neighboring Lude sites since each newly installed package is advertised to all three sites.

Collaborative development is a difficult task since each site has different priorities at different times. A volunteer must be found each time a new sub task is identified. The amount of work separating a first locally working prototype from a mature, tested and fully documented release is easily underestimated.

Three areas are currently getting our attention regarding future developments. One area is graphical user interfaces. A graphical front end could list the available software packages, modifications and classes on which operations such as copy and link can be applied. A second area is automating the software selection and updating process. A tool could list available software packages sorted by compilation date and/or keywords and even automatically perform the installation based on those criterions. Another possibility is to initially install packages through symbolic links and then based on usage decide which package should be copied locally.

A third area is security. Copying executable files always carries a certain risk. Secure communications through authentication or cryptographic checksums can be used to alleviate the risk.

Author Information

Michel Dagenais received his B. Ing. from Ecole Polytechnique de Montréal in 1983 and his Ph.D. from McGill University in 1987, both in Electrical Engineering. He is a professor in the department of Electrical and Computer Engineering at Ecole Polytechnique de Montréal. His interests include distributed object oriented programming for CAD applications, and system administration. He can be reached at dagenais@vlsi.polymtl.ca.

Stéphane Boucher graduated from Ecole Polytechnique de Montréal with a B. Ing. in computer engineering. He worked as software developer and then as network analyst for the department of Electrical and Computer Engineering of Ecole Polytechnique de Montréal. His work on Lude was completed while he was at Ecole Polytechnique. He is now a software engineer for Bell-Northern Research in Ottawa, Canada. His interests range from operating systems and compilers to Software Engineering. He can be reached at sbo@bnr.ca.

Pierre Laplante is a senior system administrator at the Centre de Recherche Informatique de Montréal. He earned a B. Sc. in Computer Science from Universite de Sherbrooke. His current interests include heterogeneous system administration using UNIX and X programming, in particular developing system administration tools with perl, wafe, c and c++. He can be reached at laplante@crim.ca.

Robert Gérin-Lajoie is Chief Laboratory Manager in the Computer Science and Operational Research Department at Universite de Montréal. He received his M.Sc. in Computer Science from Universite de Montréal in 1981. His interests include large Unix systems administration tools and methodologies, and information and knowledge access tools. He can be reached at rgl@iro.umontreal.ca.

Pierre Mailhot is currently system administrator at the Computer Science and Operational Research Department of Universite de Montréal where he previously obtained B. Sc (1986) and M.Sc (1989) degrees in Computer Science and worked 3 years as programmer-analyst with the department's VLSI laboratory. His interests include system administration, operating systems, security, distributed simulation and CAD tools for VLSI. He can be reached at mailhot@IRO.UMontreal.CA.

Bibliography

[1] J. Sellens, "Software maintenance in a campus environment: The xhier approach," in *Proceedings of the USENIX Fifth Large Installation Systems Administration conference*, (San Diego, California), pp. 21-28, October 1991.

[2] K. Manheimer, B. Warsaw, S. Clark, and W. Rowe, "The depot: A framework for sharing software installation across organizational and unix platform boundaries," in *Proceedings of the USENIX Fourth Large Installation Systems Administration conference*, pp. 37-46, October 1990.

[3] W. Colyer and W. Wong, "Depot: A tool for managing software environments," in *Proceedings of the USENIX Systems Administration LISA-VI conference*, (Long Beach, California), pp. 153-162, October 1992.

[4] D. Pukatzki and J. Schumann, "Autoload: The network management system," in *Proceedings of the USENIX Systems Administration LISA-VI conference*, (Long Beach, California), pp. 97-104, October 1992.

[5] P. Anderson, "Managing program binaries in a heterogeneous unix network," in *Proceedings of the USENIX Fifth Large Installation Systems Administration conference*, (San Diego, California), pp. 1-9, October 1991.

Appendix A

To install LUDE, one needs access to the /usr/local directory, and bin, lib, include, man, info, doc, soft and server sub-directories must exist. A new sub-directory under /usr/local/soft will be created for each software package installed.

To start, the LUDE utilities and the Perl interpreter must be retrieved. The following FTP servers are accessible on the Internet: ftp.crim.ca, ftp.iro.umontreal.ca and ftp.vlsi.polymtl.ca. With FTP, it is often easier to retrieve the complete subtree for a package and then remove the unwanted binaries (for architectures not used at your site).

Here is how to proceed for installing lude and perl:

```
% cd /usr/local/soft
% ftp ftp.crim.ca
Connected to Clouso.CRIM.CA.
220 clouso FTP server (Version 2.0 Mon Apr 12 22:48:26 EDT 1993) ready.
Name (ftp.crim.ca:dagenais): ftp
331 Guest login ok, send e-mail address as password.
Password:
230-
230-This ftp daemon support tar and compress.
230-To get a directory, append ".tar" to the name of the directory.
230-To get a compress version, append ".Z" to the name.
230-
230-
230 Guest login ok, access restrictions apply.
ftp> cd lude-crim
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
X11R5
TeX-3.141
xrolo-v2p6
procmail-2.7
et3.0-alpha.1
lucid-19.3
hyperbole-3.04
wafe-0.92
cvswrapper-0.9
xntp-3.0
bibview-1.4
etgdb
perl-4.035
lude-1.6
226 Transfer complete.
859 bytes received in 0.3 seconds (2.8 Kbytes/s)
ftp> get lude-1.6.tar.Z
200 PORT command successful.
150 Opening BINARY mode data connection for /bin/tar.
226 Transfer complete.
local: lude-1.6.tar.Z remote: lude-1.6.tar.Z
ftp> get perl-4.035.tar.Z
200 PORT command successful.
150 Opening BINARY mode data connection for /bin/tar.
226 Transfer complete.
local: perl-4.035.tar.Z remote: perl-4.035.tar.Z
ftp> quit
221 Goodbye.
```

```
% zcat lude-1.6.tar.Z | tar xf -
% rm lude-1.6.tar.Z
% zcat perl-4.035.tar.Z | tar xf -
% rm perl-4.035.tar.Z
% sh
$ PERL=/usr/local/soft/perl-4.035/run/poly/sun4.1_sparc/bin/perl
$ cd /usr/local/soft/lude-1.6/run/poly_eng/sun4.1_sparc/bin
$ $PERL lude -sof perl-4.035 -mod poly -cl sun4.1_sparc -link
$ ./lude -sof lude-1.6 -mod poly -class sun4.1_sparc -link
$ exit
%
```

Then, any other software package is easily installed. Suppose that you also downloaded modula3-2.11.tar.Z in /usr/local/soft using FTP. The following commands are now sufficient to install it.

```
% zcat modula3-2.11.tar.Z | tar xf -
% rm modula3-2.11.tar.Z
% lude -sof modula3-2.11 -mod poly -class sun4.1_sparc -link
```

The Corporate Software Bank

Steven W. Lodin – Delco Electronics Corporation

ABSTRACT

The Corporate Software Bank is the implementation of an idea borne of many hours spent installing and maintaining public domain UNIX software tools and many hours spent trying to explain the process to other UNIX workstation administrators. The Corporate Software Bank is a company-wide computing resource which makes valuable public domain software available in working form avoiding the usual pitfalls of inexperience and lack of resources.

Delco Electronics Corporation is making the transition from mainframe based computing to workgroup computing utilizing UNIX client/server technology. This has created a need for many UNIX system administrators. Like any major corporation, these administrators can range in experience from someone with no public domain software experience to the guru class administrator.

A decentralized system administration model requires that each system administrator have all the skills necessary to meet their users' needs. Due to external factors, many system administrators are unfamiliar with Usenet/Internet related functions. Of these functions, retrieving, installing, and maintaining public domain software has become more requested from the user community. The Corporate Software Bank is an attempt to solve this problem by changing the answer from a lengthy explanation of the installation process to a simple answer of "retrieve a file and follow the instructions to get that program and many more already in working form".

This paper describes the motivation behind the Corporate Software Bank. It presents a site-dependent implementation discussing the problems encountered and solutions, when applicable. Finally, the usage history is presented and some future enhancements are suggested.

Environment Description

UNIX system administration at Delco Electronics Corporation (DE) is split up according to, but not necessarily limited by, functional, geographical, and business boundaries. The company is divided into strategic business units, each with its own end product or services, that spend their computing-related budgets to meet their needs. These units have three choices to meet their computing requirements: develop internal expertise, utilize an existing DE corporate resource, or utilize a General Motors Corporation (GM) resource. They may select one or more of those resources to meet their needs. Some groups have developed considerable internal talent. Other groups rely solely on a GM corporate resource. The third choice, sometimes known as Core Engineering, is the DE corporate resource of the Software/Computer Tool Development (S/CTD) department. The S/CTD department receives its funding from the other business units to create, administer, and distribute computer projects for DE, and in some cases, all of GM. The S/CTD department provides services in drafting, electronic document management, and software development and management.

A couple of years ago, the only corporate solutions for data management were IBM mainframe or DEC VAX based. Until recently, software development was done under MVS/TSO. Solutions are now being developed and installed that support the UNIX operating environment. These solutions manage corporate data for CAD, electronic documentation, and software development. In addition, other pockets of UNIX workstation use have arisen in niche areas like finite element analysis and manufacturing testing equipment. This has created a need for UNIX system administrators.

In some groups, system administration is recognized as a full time position. In others, it is just another part of the engineering duties. This has led to a wide range of UNIX system administration experience and skills. Because of the large number of different system administration groups, information dissemination can be difficult. This has been magnified by using a decentralized system administration model. See Figure 1 for some indication of the wide range of administration ratios.

The physical layout of the DE network is diverse. It is transitioning from a bridged model to a routed model. The transition started in early 1990 when DE acquired a Class B Internet Address range and installed many routers. As part of a continual upgrade process, existing bridges and repeaters are being replaced with routers. The DE network ranges from building networks of routers, concentrators, and hubs to campus networks of fiber and T1 lines to an international wide area network connected at basically every speed available.

Our link to the Internet is through a connection to the GM network infrastructure known as Infranet. Infranet was designed to connect GM sites worldwide for data sharing. Many other GM entities are on Infranet.

The GM Infranet is connected to the Internet through a firewall. This firewall went into pilot usage in December of 1992 during which there were a few select accounts. It went into production usage in July of 1993, at which point accounts were available to anyone presenting a business case.

To summarize, there is a large network infrastructure in place to support the growing community of UNIX workstations. In addition, there is a general lack of Usenet/Internet experience for system administrators. There has also been a lack of access to the Internet for a normal user or unsavvy administrator until recently. This has resulted in unfamiliarity with public domain software both from a user and administrator standpoint.

Motivation

Supporting a large engineering user community in a large heterogeneous environment has required the use of public domain software to maintain a productive work environment. In creating this environment, we feel we have made the most powerful computing environment for the users and the administrators in our company. This has created a situation where users on other systems have continually requested access to the public domain tools we had available. Some of these tools are the traditional ones like **emacs** or **elm**. Others are advanced programs like **xntp** or **amd**.

Also, some of our internally developed software depends heavily on public domain software such as **perl**. Initially, we expected the sites to which we distribute software to have the public domain software already installed, as a prerequisite. This was a mistake. After trying to explain the process of installing public domain software, we also realized this was a mistake and we started including the software in the distribution. There were no licensing issues since we do not charge a fee for our software.

Increasingly, more time was spent making public domain software available, explaining the process for compiling and installing the software, and occasionally, getting an account on a remote system and installing the software. It was realized that unless something was done that would streamline this process, the effort for public domain software would soon be unmanageable. We analyzed the situation and came up with the following possible problems that needed to be addressed:

- Make programs available in working form
- Install software consistently and securely
- Make sources available
- Support multiple architectures
- Reduce disk space utilization

Based on the range of experience of the system administrators we had been dealing with, the solution would have to be a quick step-by-step method. The solution to these problems is the Corporate Software Bank (CSB).

Making programs available in working form would save time because the program would be installed once, in one location. The build and installation process would not need to be explained many times to many different administrators. In many previous instances, users would get their system administrators to call to get a program installed on their machine. The users and system administrators would expect something simple like copying over one executable. Most of the time, they did not feel the effort was worth the benefits of the program and the user needed to find alternative methods for solving his problem. To be fair, for someone who has never installed public domain software, it can be daunting. Once you do it a couple of times, it's just like following a recipe. The **configure** program included with GNU packages and the usage of **imake** in X11 programs has made the build process of public domain software much easier.

It was considered important to have a consistent methodology for compiling and installing the public domain software while still making it easily

Site – Admins	Seats	Accounts	Active Users
Site A – 4	250	830	500
Site B – 5	56	150	60
Site C – 2.5	87	122	80
Site D – 1	3	66	26
Site E – <1	4	60	40
Site F – 4	275	360	60
Site G – 2	40	550	450

Figure 1: Administration ratios

available to others. This implied a single hierarchy for installation instead of installing programs all over the filesystem like **/usr/bin/X11** or **/usr/local/bin**, or having to install in **/package_name** and using links. Another concern was to have a methodology for installing the software in a fashion that eliminates any security risks. The traditional guidelines for checking out public domain software are followed.

Making sources available is not as important now that there is an Internet connection that is accessible by everyone. It is nice, however, to have a single location known to have up to date sources on the local network. In addition, for those administrators that wish to have certain software running locally, this allows them to utilize the CSB for testing the software, then later copy the software and compile and install it locally.

Supporting multiple architectures is a problem that the CSB tries to solve to a limited extent. The list of currently supported platforms happens to coincide with the list of platforms supported by our system administration group. However, all software that is installed in the CSB for UNIX platforms is installed for as many of those platforms as possible. Obviously, some public domain software is not built to be compatible on all platforms, and it is not worth our time to try porting it. Generally, the more popular (which is usually proportionate to useful) a piece of software is, the more platforms it is supported on.

Reducing disk space utilization was a consideration for implementation that is based on our administration model. Our workstations are set up as dataless so the only uses of the local hard disk on a workstation are operating system, windowing system, and swap space [Nelson92]. As a result, we did not want to have a solution that required adding software to the local workstations. Also, having a good network infrastructure for the DE Kokomo campus allowed us to decide to make the CSB available to everyone else utilizing the network.

One of the other problems that we encountered was having multiple copies of sources and executables scattered both in system and user directories. Granted, this problem could have been worse if we had an Internet connection that all users had access to. Having one central location has helped eliminate this problem. Users now know that there is a location and methodology for making public domain software available to them.

Attempting to solve those problems would hopefully provide the following benefits:

- Manpower savings
- Enhanced productivity
- Material savings
- Quality improvement

It saves the installer time by only requiring the installation of the software once, not once for our workstations and once for each of the other workstations that want the program. It also saves time for all the other administrators because they do not have to individually install the software. Once they do the initial installation of the CSB, they have access to all the programs.

In the end, the user benefits from enhanced productivity because high quality software is available to them at little or no cost in terms of disk space or administrative time. It also gives users working at remote sites access to the same tools as the home site.

The implementation and the network infrastructure allow disk space savings to be realized since the CSB can be accessed across the network. This may not seem like much at first, but when you start to consider the tens to hundreds of megabytes on hundreds of workstations then it starts to add up.

Finally, there is an implicit quality improvement. In most cases, the most recent version of a particular piece of software is available. For public domain software, a newer version usually includes bug fixes and program feature upgrades. In the case where a vendor delivers the software with the operating system, like HP with elm, the CSB offers a newer enhanced version.

While working on the initial concept, it was decided to try to make this as general as possible to appeal to as many users or systems as possible. This meant that no proprietary or strategic data or proprietary programs would be installed or accessible. This also meant that our group could not install some of the locally developed programs that were designed for our own user community. Also, based on the administrative boundaries and funding policies, it would be impossible for commercially licensed software, such as Lotus 123 or Interleaf, to be made available via the CSB. At this point, it was decided that only public domain software and locally developed software that could be distributed freely should be installed in the CSB. Finally, it was decided that the CSB would not be an infinite storage facility. This affected the implementation in two ways, the clients would mount the CSB read-only, and programs would use the local workstation disk for temporary files. This way, no programs would be writing to the CSB disk, possibly filling up the filesystem.

Implementation

The implementation of the CSB is highly site dependent. An explanation of the reasons behind our implementation is presented. The first thing noticeable is that we did not use **/usr/local** as the hierarchy. The basic reason for this is that in our client workstation model, **/usr/local** is on the local hard disk of the workstation and contains only the software required to be present when the network is not. In addition, we realized that other groups have been using **/usr/local** for their own purposes and we did not want to cause any conflicts if possible. This meant that we would have to create a new directory structure. We chose **/usr/std** for historical reasons. The choice of std actually worked out well, because users now reference the CSB as "standard" which can be construed as the meaning for std.

As illustrated in Figure 2, the layout of the CSB is based on architecture. Each architecture has its own hierarchy similar in layout to a normal /usr/local hierarchy. Each client workstation mounts the filesystem from the server at the architecture mount point to **/usr/std**. The development workstations have the ability to mount the CSB one level higher to have access to all architectures and the source directory in addition to mounting as a client workstation for normal access to the CSB. The dashed lines in Figure 2 show the NFS mounts that client and development workstations perform to gain access to the CSB. In a heterogeneous environment, the use of an automounter that can key on the workstation architecture facilitates the mounting process immensely.

The server utilized in our implementation is a Auspex NetServer NS5000. The Auspex server is the backbone of our dataless client administration model. As an added benefit, it has performed flawlessly in its role to serve large amounts of data to many clients. The important characteristics of this server that make it suitable for the CSB are:

- Industry leading NFS throughput
- Multiple Ethernet connectivity
- Large disk capacity
- Filesystems larger than 2GB

As demonstrated in the latest SPEC SFS Release 1.0 Benchmark Suite (097.LADDIS), Auspex is leading the pack in terms of NFS throughput. In particular, our Auspex is a hybrid between the NS5000 and NS5500. It is running version 1.5 of the Auspex operating system, has 23 SCSI drives ranging in size from 1GB to 2GB for a total usable disk space of 37GB, and has the maximum of 8 ethernets.

The Auspex server is capable of disk striping, concatenation, and mirroring. We have created a 5GB virtual partition that is striped across 5 disk drives for the CSB and other network available software. We limited the partition size to 5GB so it would fit conveniently on a Exabyte-8500 written tape.

With its 8 Ethernet interfaces, the Auspex server appears locally on 8 different subnets. This allows us to bypass a router, making the delivery of data even faster. Over 450 workstations and PCs have direct access to the server.

The current implementation of the CSB only supports hp300/hp400, hp700, and sun4 architectures. In addition, the sun4 architecture is limited to the least common denominator, SunOS 4.1.2. The supported architectures coincide with the architectures supported by our system administration group. We do not have access to the other platforms at DE such as Apollo Domain, Intergraph, and DEC Ultrix, and volunteers have not been recruited yet. Also included in the CSB are directories for PC-compatible and Apple Macintosh programs. The programs included for those architectures are network applications and UNIX-like utilities.

To save disk space, hard links are used between architectures where possible. This includes library files, info files, and manual pages. See Appendix A for the CSB client installation instructions and checklist.

It was a requirement that simple instructions were to be provided in order to make the CSB available to everybody. The client installation instructions are listed in Appendix B. It is a seven step process that needs to be done as root on the CSB client machine.

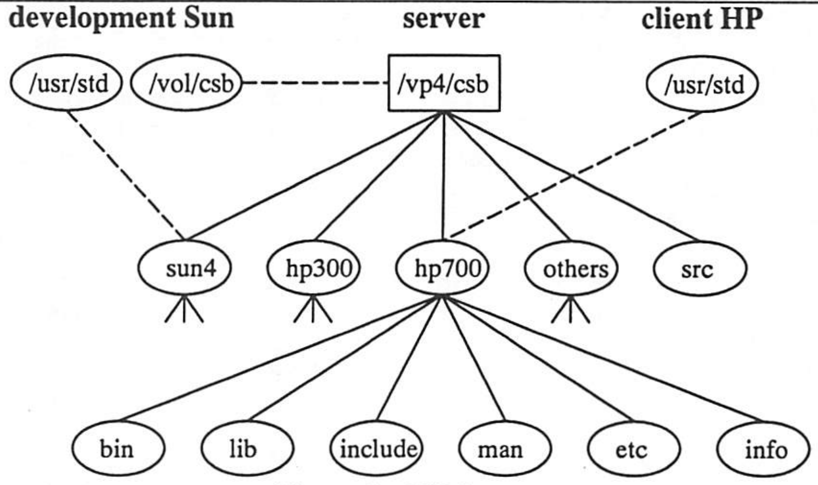

Figure 2: CSB layout

In the instructions, it is specified that the clients mount the CSB read-only. Since we did not want to control who had access, everyone gets access. We did not want to leave the CSB open for attacks of denial of service by filling up the disk drive, or becoming a privileged user and being mischievous. We also did not want to maintain a large **/etc/exports** file or NIS netgroup. The server has a simple exports line:

```
/vp4/csb -rw=kocrsv01:kocrsw15,
root=kocrsv01:kocrsw15:,anon=97
```

This gives read-write access only to the development machines and maps anonymous uids to the csb uid. The five development machines are sun4 workstations with SunOS 4.1.2 and SunOS 4.1.3, an hp300 workstation with HP-UX 8.07, and hp700 workstations with HP-UX 8.07 and HP-UX 9.01. We used [Stern91] and [Garfinkel91] as references.

Problems

During the implementation and the throughout its usage, there have been problems associated with the CSB, both technical and non-technical. The biggest non-technical problem has been the acceptance of the CSB. It has not be as widely accepted as possible because of the negative image some people (unfairly) associate with public domain software. This image usually arises due to fear of the unknown, the virus scare, and the licensing issues with shareware.

The other non-technical problem is the acceptance by other system administrators to use a resource not developed by themselves or not commercially supported. This attitude is difficult to change. In this situation, the merit of the software can do more persuading than words.

Another problem was spreading the word to all the users and system administrators that might benefit. Since there is no central system administration group and there are many different organizations using UNIX workstations, this was difficult. Notices were sent out to the members of the DE UNIX Users Group mailing list and to the system and network administrators in the DE Network Users Group. In addition, some GM users were introduced to it during a Software Tools User Group meeting. Finally, other GM users posting to NetNews asking about public domain software were sent the CSB information since they can access the CSB on the GM network.

Some software problems have been encountered. To reduce the security risks, we have not allowed any programs to be installed setuid/setgid. This means programs like the HP monitor program (which needs setgid to the kernel group to access kernel data structures) will not work when run as a normal user. It must be run as root on the client workstation to have access to the kernel.

Another problem was X11 programs in general. Most X11 programs like to be installed in the default X11 binary directory **/usr/bin/X11** and they like to have their application resource files installed in **/usr/lib/X11/app-defaults** along with their fonts in **/usr/lib/X11/fonts**. The standard X11 startup programs default to those locations. Since we wanted the CSB to be a standalone hierarchy, it was decided that installed X11 programs would have their application resources stored in **/usr/std/lib/app-defaults** and their fonts in **/usr/std/lib/fonts**. This paralleled the standard configuration of **/usr/lib/X11**. All our users have the XAPPLRESDIR environment variable pointing to a directory in their user area called **.app-defaults**. It took a while to figure out that the XAPPLRESDIR environment variable is like the PATH or MANPATH variable in that it can have multiple entries colon separated. Before that discovery we advised the users to copy the application resource files out of the **/usr/std/lib/app-defaults** directory into their **.app-defaults** directory. The X11 fonts in the CSB can be added with an **xset** command.

Most public domain software available today seems to have been developed on a Sun or some machine running a BSD-derivative operating system. It will be interesting to see if this changes now that Sun delivers Solaris 2.x, a SysV-based operating system. Anyway, compiling software on HP-UX is not as easy as generic BSD or generic SysV since it seems to have a SysV kernel with BSD extensions. Of great help was the "INFO: GNU products on HP9000s700" information posted regularly to comp.sys.hp by Pierre Mathieu which lists the GNU programs that compile without modification and gives the modifications for those that need it to compile on HP-UX. In addition, the "BSD to HP-UX porting tricks" article posted regularly to the newsgroup comp.sys.hp by Mike Peterson has helped out. Finally, for software guaranteed to run on HP workstations, the HP Interworks organization has an anonymous ftp site, iworks.ecn.uiowa.edu, and the Liverpool UK HP-UX Software And Porting Archive organization has an anonymous ftp site, ftp.csc.liv.ac.uk. These two generous groups post updates on their software libraries regularly on the newsgroup comp.sys.hp.

There are three connectivity models for accessing the CSB:

- high speed network access – NFS
- low speed network access – **rdist**
- no network access – tape

The CSB was designed to perform best using a high speed network infrastructure. In testing, some software would fail at remote sites connected at 56KB. Other software would take excruciatingly long. Based on this, it was decided that anything connected at 56KB and less should install the CSB locally. This is solved relatively easily using **rdist**.

The instructions for remote site installation and the **rdist** distfile are shown in Appendix C.

Not every site we work with is connected to our network or Infranet. In those cases, we have distributed the CSB via tape. We can accommodate a wide variety of tape media. This is the worst case scenario.

There have been a couple of other miscellaneous problems encountered. Due to having workstations not administered by us and unknown to us mounting the CSB, the exported mount point is relatively permanent. This is only a minor inconvenience. Recently, when we upgraded the filesystem from 2GB to 5GB, it required a little extra work.

Another problem is one of unauthorized local modifications. For example, a program is installed correctly in the CSB but does not work quite right on a workstation, possibly due to an administration or configuration deficiency. The program is copied locally and modified to work around the problem. When we send out an upgrade, the copied program mysteriously quits working. This is difficult to track down and fix.

A relatively minor problem with HP-UX is that it only allows for one whatis manual database. It will now follow the MANPATH environment variable, but it requires a single merged database in **/usr/lib/whatis**. This is described in the HP-UX *catman* (1M) man page.

Upgrading existing packages in the CSB is currently done randomly. There is no established procedure. Upgrades are done when users or administrators notice that programs have new versions. Typical notification comes from the announce newsgroups or the source newsgroups and from mailing lists. This is one area that needs to be improved.

Usage History

There are three statistics that are considered important, the number of packages installed, the amount of disk space utilized, and the number of machines mounting the filesystem. These statistics are presented for information only, not for numerical analysis.

The first statistic is the number of packages installed. This number should be increasing to keep the resource from becoming stale. In addition, another important factor is keeping the existing packages up to date. When the CSB was announced to the user community in April, it contained over 50 packages installed for the supported UNIX platforms. In August, it contained over 60 packages for the UNIX platforms. See Figure 3 for more information. Of the packages installed, 35% have been upgraded.

The second statistic that is considered important is the amount of disk space utilized. There are 2 categories of utilization, source and working form. Figure 4 shows the change in utilization for the three binary architectures supported [hp300, hp700, sun4] and for the sources [src]. Due to their insignificance, we have neglected to show the utilization for the PC and Mac directories. The reason for the reduction in the src directory is that the source directory is also the build location and at any given time, some sources may be uncompressed and untarred for maintenance or installation. The amount of disk space utilized is important for remote site distribution.

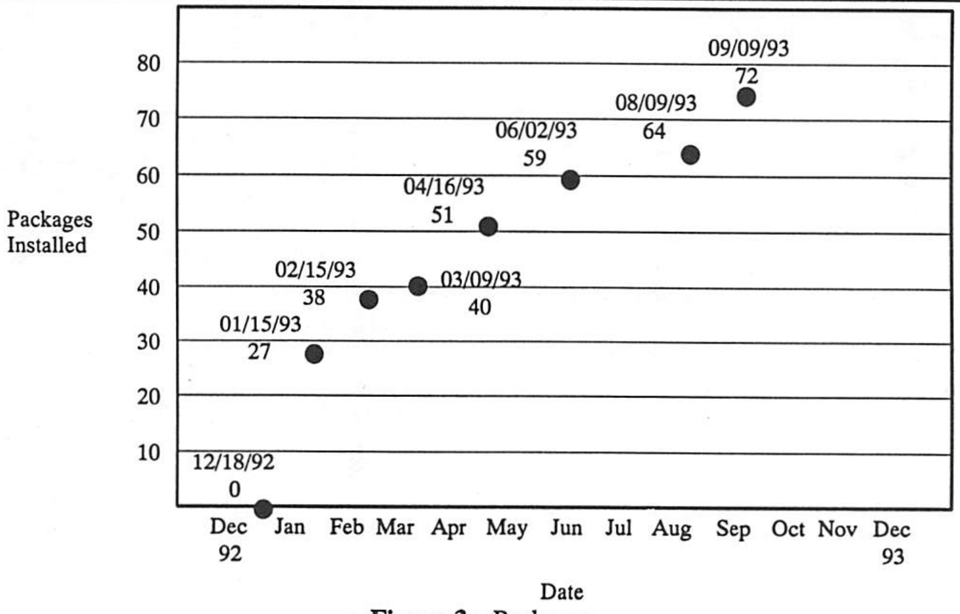

Figure 3: Packages

The number of machines utilizing the CSB is the third important statistic. This will show the number of machines actually using the CSB at a given time. Of the over 200 workstations that we administer, there may only be a fraction that have the CSB mounted due to the automounter. The ratio of our workstations to the total amount of workstations mounting the CSB is generally around 70%. See Figure 5 for a couple of data points, taken at random, of clients mounting the CSB across the network. This data was gathered by doing

```
grep csb /etc/rmtab | grep -v ^# | \
             uniq -u | wc -l
```

on the Auspex server. The Auspex at any given time is supporting over 6300 active NFS mounts from over 300 clients.

Future Enhancements

To most improve the CSB, the available list of software and if possible, the supported platforms must continue to grow. The available list of software needs to be increased by adding different binaries and making more sources available. In particular, including both the full X11R5 and GNU sources would be beneficial. As always, disk space is a consideration along with download time and

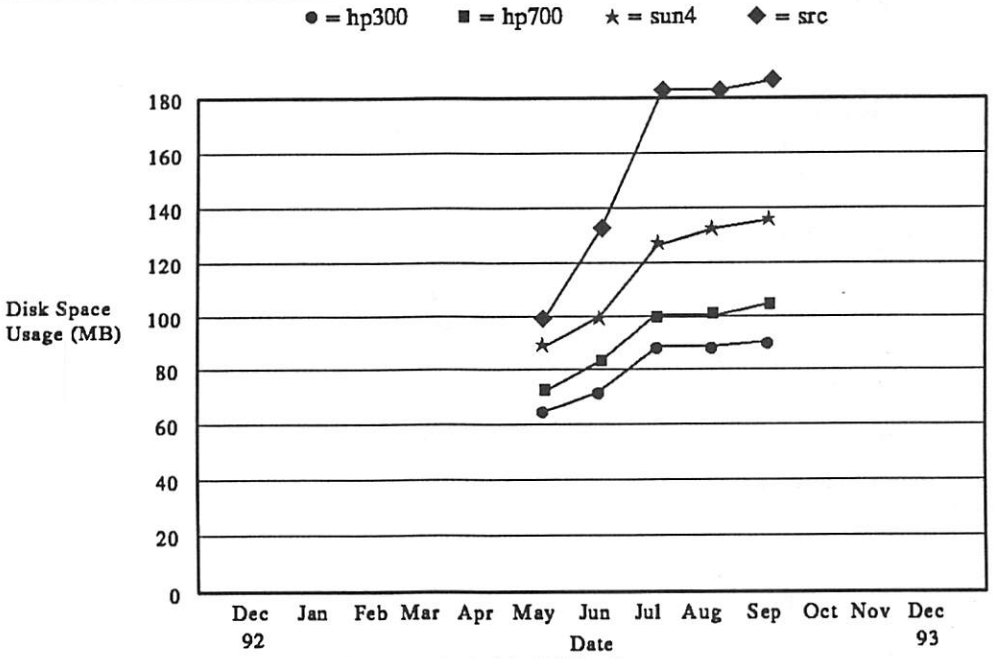

Figure 4: Disk Utilization

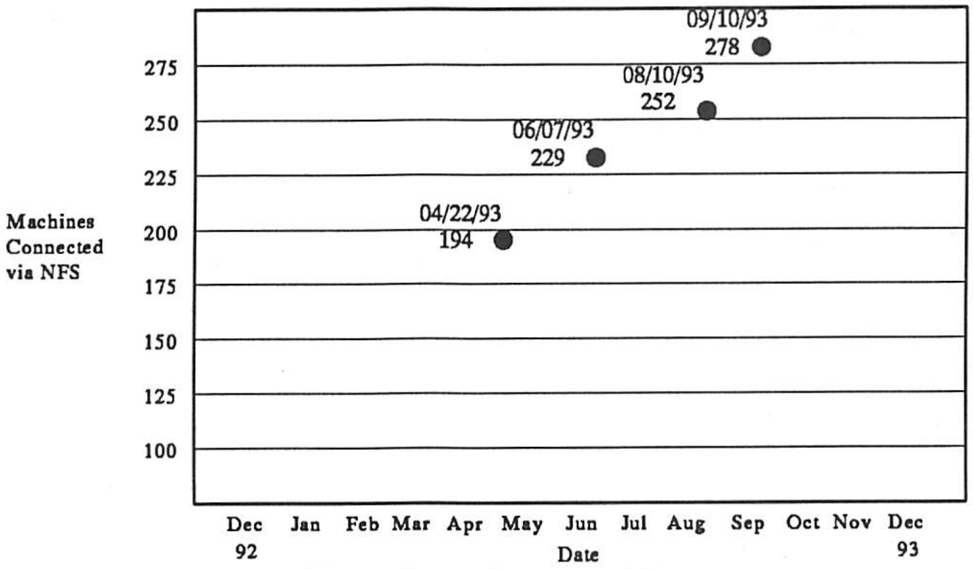

Figure 5: Machines using CSB

network bandwidth. In addition, more PC and Mac sources related to networking, email, and NetNews would be beneficial. The problem with those is finding someone knowledgeable and experienced to support them.

One possible solution to the diskspace dilemma is to utilize a CD jukebox facility to make X11R5, GNU, and other public domain software CDs available. Other possible CDs include the NetNews CD and Sun's Catalyst CDWare CD. This solution has not been explored further.

Another possible enhancement for the CSB would be to create and maintain a vendor patch directory. This would make vendor patches available to all the different system administrators. Since our primary vendors, Sun and HP, have patches available on the Internet, it would be relatively simple to start the patch directory. However, the difficult part would be maintaining it. This would include keeping separate directories for different operating system releases. Another maintenance task would be to remove patches that are superseded by newer patches. Since manpower is limited, this enhancement probably will not be implemented in its full capacity.

Although this does not seem feasible currently for a number of reasons, it would make sense to load any site licensed software and make it accessible to everyone via the CSB.

One of the more time consuming future activities will be creating a Solaris 2.x architecture. Since we do not anticipate supporting that architecture until the third quarter of 1994, we have plenty of time to prepare for it. Some software already has been ported; others will compile fine using existing SysV parameters. We will keep watching the net for hints.

Conclusion

The Corporate Software Bank is a valuable resource at Delco Electronics Corporation. It has saved both time and money. It is available on the DE Kokomo campus via NFS, on the DE wide area network or the GM wide area network via weekly rdists or to our unconnected customers via magnetic media. There is a README file describing the benefits, installation, structure, and installed packages available from the author via email or via anonymous ftp on the GM wide area network at kocrsv01.delcoelect.com in the doc directory.

The Corporate Software Bank is an evolving resource. It is continually being improved. All suggestions for improvement will be accepted. Please send them to the author.

Acknowledgments

Many thanks to the other Software/Computer Tool Development system administrators, Chris Cleary (DE), Ryland Rusch (DE), and Karen Kerr (DE). Thanks also to the CSB supporters, Dianne Glickfield (EDS), Mike Thoeny (DE) and Mike Scheidler (DE). Finally, many, many thanks to Rob Kolstad (BSDI), who at a critical period in my life, kept me on the system administration career path.

Author Information

Steven W. Lodin earned a B.S. in Electrical and Electronics Engineering with Computer Option from North Dakota State University in 1988. He has worked for General Motors since 1985. Currently he is working in the Software/Computer Tool Department of Delco Electronics Corp., which is a subsidiary of GM Hughes Electronics, which is a wholly-owned subsidiary of General Motors Corporation. As part of the System Administration Group, he is responsible for the administration of a heterogeneous world-wide network utilized for Systems Engineering and Software Development for most of Delco Electronics vehicle electronic systems. He is currently spending most of his time working with NetNews, email and of course, public domain software. Reach him via US mail at Delco Electronics Corp., MS CT-LL-M, One Corporate Center, Kokomo, IN 46904-9005 or send mail electronically to swlodin@kocrsv01.delcoelect.com.

References

[Garfinkel91] Simpson Garfinkel and Gene Spafford, "Practical UNIX Security", O'Reilly & Associates, June 1991.

[Harrison92] Helen Harrison, "So Many Workstations, So Little Time", LISA VI Proceedings, pp. 79-87, Long Beach, October 1992

[Nelson92] Bruce Nelson, Rapheal Frommer, & Auspex Engineering, "An Overview of Functional Multiprocessing for NFS Network Servers", Auspex Systems, August 1992.

[Schafer92] Peg Schafer, "bbn-public -- Contributions from the User Community", LISA VI Proceedings, pp. 211-213, Long Beach, October 1992

[Stern91] Hal Stern, "Managing NFS and NIS", O'Reilly & Associates, June 1991.

Appendix A: Instructions for Package Installation (v3.0 09/01/93)

1) Put the source package in /vol/csb/src.
2) Unpack the package.
   - If the package ends in .tar.Z then uncompress and untar.
   - If the package ends in .tar.gz or .tar.z then gunzip and untar.

- If the package ends in .sh or .shar, the unshar it using sh.

3) Read the README and INSTALL files.
4) Change package configuration options:
   - Modify and config.h or conf.h files per instructions.
   - Modify Makefile per instructions.
5) Install the package:
   - If GNU-type package then
     - modify Makefile.in to change the prefix to be /usr/std from /usr/local
     - configure
     - make
   - If X11 package then modify the Imakefile to include:
     BINDIR = /usr/std/bin
     LIBDIR = /usr/std/lib
     XAPPLOADDIR = /usr/std/lib/app-defaults
     EXTRA_LIBRARIES = -lm
     Then

```
xmkmf
make
```

   Then
     strip the executables
     make install
     remove any library or man pages installed
     link libraries and man pages
     make clean
   - Otherwise, follow the instructions in the README or Makefile.
6) Repeat step 5 for all platforms applicable.
7) Save the package with the modified configuration:
   Change directory to parent above the package.
   Tar and compress the package:

```
tar cf package.tar package
compress package.tar
```

8) Remove the package build directory.

```
rm -rf package
```

9) Add entry to the README file. Increment the minor release number at the top. Copy the new README into the anonymous ftp directory on kocrsv01.delcoelect.com (~ftp/sys_info/CSB.README).
10) Post to NetNews in the delco.software.csb newsgroup. Post man pages sections if possible. If a minor upgrade, post the CHANGES files.

Appendix B: CSB Installation

Now that you know more about the CSB and you've decided that you want to start using it, there are some fairly simple steps (executed as root) that need to be taken. I am going to illustrate a single architecture mount example for the HP700 platform. This assumes the workstation is configured properly running DNS (Domain Naming System) or has a current hosts file. The most up-to-date hosts file can be retrieved from kocrsv01.delcoelect.com via anonymous ftp in the sys_info directory.

- Outside CTC Building
  1. Create directory /usr/std (mkdir /usr/std)
  2. Modify /etc/checklist to add the following line:
     kocrsv04:/vp4/csb/hp700 /usr/std nfs ro,bg,soft,intr 0 0
  3. Mount the directory (mount /usr/std)
  4. Modify the default or each user's PATH environment variable to include /usr/std/bin
  5. Modify the default or each user's MANPATH environment variable to include /usr/std/man
  6. Modify the default or each user's XAPPLRESDIR environment variable to include /usr/std/lib/app-defaults
- Inside CTC Building
  Modify the line in /etc/checklist or /etc/fstab to use one of the following instead of kocrsv04. This is because the CSB server has multiple ethernet ports which allow direct access to some floors/wings bypassing the CTC router.

If on floor	Use	If on floor	Use
CTC-LL	sv04_02	CTC-1E	sv04_04
CTC-2E	sv04_06	CTC-3E	sv04_08
CTC-3W	sv04_09	CTC-4E	sv04_10
CTC-4W	sv04_11		

Appendix C: Weekly Rdist Installation Instructions

This is the suggested method for access to the CSB from WAN connections (DE WAN or Infranet). The binaries, libraries, and man pages are all available locally, with weekly updates via the rdist command.

Required:

- The rdist command in /usr/ucb/rdist. We can provide this for HP workstations since its not included in HP-UX.
- Enough disk space to hold the CSB for each architecture requested. This is a constantly increasing number. Check with the CSB administrator for the latest guess.
- A local account, csb, on the local CSB server. It should not be in NIS. A suggested /etc/passwd entry is:

```
csb:*:97:9:Corporate Software Bank,,,:
     /vol/csb:/bin/csh
```

- It will also need a .rhosts file in its home directory with the following machine entries (these are all for just one machine):

```
kocrsv04
kocrsv04.delcoelect.com
sv04_03
sv04_03.delcoelect.com
```

Rdist file

```
REMOTE_CSB_ALL = ( mwbchp01 )
REMOTE_CSB_SUN4 = ( gmpxhp02 edssun ele1 lxeusv01 sun01 )
REMOTE_CSB_HP700 = ( flidh102 gmpxhp02 edssun sun01 hp22 )
REMOTE_CSB_HP300 = ( flidh102 )

inst:
        (/vol/csb/sun4 /vol/csb/hp700) ->
        (sun01) install;

csb.all:
        (/vol/csb/sun4 /vol/csb/hp700 /vol/csb/hp300) ->
                (${REMOTE_CSB_ALL}) install -R;

csb.sun4:
        (/vol/csb/sun4) ->
                (${REMOTE_CSB_SUN4}) install -R;

csb.hp700:
        (/vol/csb/hp700) ->
                (${REMOTE_CSB_HP700}) install -R;

csb.hp300:
        (/vol/csb/hp300) ->
                (${REMOTE_CSB_HP300}) install -R;
```

Customization in a UNIX Computing Environment

Craig E. Wills, Kirstin Cadwell, & William Marrs – Worcester Polytechnic Institute

ABSTRACT

This work studies the use of customization in a campus UNIX computing environment where a large computing culture exists, but one that has many diverse interests rather than focusing on a cohesive project. We found there to be a gap between the core system users who use customization facilities and periphery users who show an interest in customization, but are unaware of how to pursue it. These results point to the need for better means for connecting novice users with useful customizations.

Introduction

Customization of computing environments is increasingly available to computer users, but its use often does not match its availability. This work looks at the use and sharing of customization in a campus environment. The setting is the campus of Worcester Polytechnic Institute, which is predominately an engineering school with approximately 2500 undergraduate and 1000 graduate students. Most of the students and faculty have computer accounts on the central computing facilities of the campus, which provides access to a UNIX [9] computing environment and many workstations running the X Window System [10].

In this environment there is much collaboration and sharing by students. Electronic communication and bulletin boards are frequently used. Common assignments often find students in communication either via or around computers. Students work on small-team group projects as part of their courses and explicit project requirements. The result is a ''computing culture,'' which promotes interaction and sharing of information, particularly among students. Part of this interaction involves how users design their computing environment. By the nature of the environment, students can customize aspects of the command interpreter, window environments, and specific applications such as editors and mailers. Each of these customizations is influenced both by individual preferences and the computing culture as a whole.

Previous work on customization has observed that customization of a user's computing environment is not always a solitary task, but is also influenced by the particular computing culture [4,6]. These studies have identified small groups of users within the computing environment who are responsible for translating and communicating customizations among the community as a whole. In [6], this person is termed a ''handyman'' referring to his or her ability to bridge the gap between workers and computer professionals. Mackay [4] concludes from her work that the design of customizable software should provide:

- the ability to browse through others' useful ideas,
- better mechanisms for sharing customizations,
- methods of finding out which customizations are used and effective, and
- methods of identifying customizations that are ineffective.

Our work explores how users customize their computing environment and how customization techniques can be made more visible to, and therefore used by, more users in a campus environment [12]. This work is motivated by two factors. First, which we emphasize in this paper, to study the use and sharing of customization in a campus setting where a large and diverse computing culture exists. Second, to explore the development of a tool to facilitate sharing and make customization features more accessible and understandable to novice users. Not only should this tool provide for current customizations, but more importantly be extensible so new customizations can easily be added. This tool directly addresses the needs identified by Mackay; serving as a database of useful customizations that can easily be incorporated into the user's environment.

This work is important because it examines a larger and less cohesive computing environment than previous studies. On the order of 3000 users are studied in some depth concerning their use of customization features. 13% of the users had never logged in to their accounts, while 57% had logged in during the last two weeks and 72% in the past two months. Specific customization uses are collected from 224 users and 13 are interviewed in detail. The result of this project is to learn about the culture of customization in a campus UNIX computing environment.

This paper presents the results of our work and its implications. It begins with a look at related work and the computing domain here at WPI. We go on to discuss the study of customization that was done and the results that were obtained. Based on these results, a customization tool, *ctool*, was constructed as an initial attempt to facilitate sharing of customization. We conclude with our experience with it, a discussion of future directions of this work and a summary of our findings.

Related Work

The UNIX system is powerful, but often not a friendly environment for new users. Early on, Norman [7] discussed many faults of the UNIX human interface, but it has still become a common environment in academic and industrial settings, providing many opportunities for customization.

MacLean, et al. [6], emphasize the idea that tailorability of a system requires two pieces: ''a system that can be tailored'' and ''a culture within which users feel in control of the system and in which tailoring is the norm.'' They define a ''Tailorability Mountain and its Inhabitants'' model where the worker lives on the plains, the tinkerer on the foothills and the programmer on the peaks. The handyman is a person who lives on both the peaks and the foothills to bridge the gap between workers and programmers.

Related to this work is a paper by Hesketh that also uses a button-based interface called Perly [3]. His work with the UNIX interface involves the use of ''Perly'' buttons, on-screen graphical buttons that connect to UNIX command scripts and can be shared between users.

Mackay concentrates on the influence of the community in the customization of individual systems. In [4] and [5] she presents results from interviews of 51 people concerning customization in a project environment. In looking at triggers and barriers to customization she found two of the primary barriers were that the system seemed too hard to modify and a lack of time.

Anothering interesting customization approach is using one application, a *customizer*, to dynamically change the appearance of another [8]. The customizer application provides users a common mechanism to tailor other applications.

Computing Domain

WPI is predominately an engineering school with approximately 2500 undergraduate, 1000 graduate students and 300 faculty. At the undergraduate level there are approximately 200 Computer Science majors with the remaining students in engineering, science, math and management fields. All students, faculty and staff on campus may obtain user accounts on machines of the College Computer Center (CCC). The computer center provides over 3000 accounts. Approximately 75% of the accounts belong to undergraduate students, with the remainder split between graduate students, faculty and staff. Nearly all undergraduates and faculty have accounts on the central computing system.

At the time of the study, the principal CCC machine for computing was an Encore Multimax running the UNIX operating system, with many DEC color workstations, running the UNIX system and the X Window system, also widely used. Other machines exist within specific departments on campus, but most computing needs are supplied by the CCC, particularly for undergraduate students. The large coverage of WPI users by the CCC machines makes it an attractive environment to study the characteristics of a campus computing environment.

As a basis for studying the use and sharing of customization techniques in our campus environment we identified specific applications that allow customization and are widely used. The most visible application to UNIX system users is the command interpreter or shell [1]. The predominate shells used on campus are *csh* and *tcsh*. Applications in the UNIX environments are customized primarily through ''.files'' (dot files), which are stored in a user's home directory and normally not seen when a user lists files. Customization of the shell is controlled by the files created. A default copy of each file is placed in a new user's directory as part of account creation.

The X Window System is run on each of the DEC workstations. No default customization files exist for the window system, but two files can be used to tailor the environment. The .Xdefaults file controls such window aspects as the screen layout, colors, window size, and fonts. This file is automatically created if the user invokes the DECstation session manager to control the X Window environment. The .X11Startup file is executed each time the user logs into a workstation. It is primarily used to initially create windows for running applications such as a clock, load monitors and remote logins.

Electronic mail is an important application used by the majority of users. Many mailers are used with each having their own customization files for setting up features such as mail aliases. The most common mailer is the UNIX *mail* program, which is controlled by entries in a .mailrc file. One other application studied in our work is the editor. The most widely used editor is GNU *emacs* from the Free Software Foundation [2,11]. It is both customizable and extensible, with settings controlled by the file .emacs.

The Study

Given that these applications are widely used and can be customized, our primary interest was how they are being used in a campus environment. To determine student use of customization possibilities, we created two data collection programs. These programs – one voluntary, one run on every account

– collected data on specific aspects of customization. The first collected customizations that would be useful sharing with other users. It also allowed us to find heavy users of customization who could be subsequently interviewed to gain insight into their customization habits.

The second data collection program was run system wide on each user's account and used to obtain an overview of the customization level of the users on the WPI system. From these data we were able to determine percentages of users at specific customization levels. Based on our experience and previous studies we expected to find a low level of customization.

The first program was a shell script and was voluntarily run by 224 users on the system who responded to requests for help on the project. It gathered specific information from shell, X window, *mail*, and *emacs* dot files. The program maintained a file of all user shell aliases and collected a list of all user dot files. It also asked users if they would be willing to participate in a follow-up interview.

The second program was also a shell script, but it was run by the system administrator on each user account on the system. To protect user privacy, nothing was copied from user accounts and all data were stored without identifying the user. The script searched for counts of information such as aliases, whether a user's shell dot files were the defaults and a list of dot files for the user. Data were collected on 3216 users.

The Results

Given the large number of user accounts that were studied, the first task was to gauge the level of account activity. The system-wide data collection program collected the last login time of users revealing that 415 (13%) of the users had never logged in to their accounts. However, 1834 (57%) had logged in during the last two weeks, and half of the remaining 30% had logged in during the previous two months. These figures indicate a high degree of system activity with the number of inactive accounts constituting a minority.

Customization Files

To measure the customization levels of the users, the program analyzed dot files of applications under study. Table 1 shows the level of customization on the dot files for the shell, the UNIX *mail* program, *emacs* and the X Window System. It shows that approximately 45% of the users have not modified at least* one of the default shell configuration files. For the other applications, which have no default files in a new user's account, 15-25% of the users had created customization files. However for the .Xdefaults file, half of the users had done so. Later investigation found that the use of color, provided by the DEC session manager and stored in the .Xdefaults file, was often the first level of customization.

The data collection program also gathered information about additional dot files shown in Table 2. The .newsrc file controls what network news a reader is interested in. As the number indicates, this application is also used frequently by our computing community. The .plan file is used to store personal information that can be accessed by other users. The .friends and .enemies files record other users the person is interested in highlighting when using a locally written piece of software to display where users are logged in. The table shows most users use the default DECstation window manager as opposed to alternative window managers such as *twm*.

File	Ave # of Lines	# found
.newsrc	731.2	1309 (40%)
.friends	24.7	532 (17%)
.plan	13.6	489 (15%)
.enemies	2.4	249 (8%)
.twmrc, et al.	211.5	173 (5%)

Table 2: Other Common Dot Files

Shell Customizations

The system data collection program collected many statistics about use of various shell constructs. The shell has its own programming language and use of these constructs indicate a higher level of customization sophistication. The data collection program collected usage for the following constructs, which

File (Default # of Lines)	Ave # of Lines	# Default	# not exist
.login (9)	12.4	1409 (44%)	8 (0%)
.cshrc (6)	12.6	1453 (45%)	17 (1%)
.mailrc	7.9	N/A	2489 (77%)
.emacs	20.0	N/A	2448 (76%)
.Xdefaults	35.6	N/A	1597 (50%)
.X11Startup	5.9	N/A	2668 (83%)

Table 1: Dot Files Analyzed

are used in shell dot files: aliases, set, setenv, stty, echo, comments, if, and back quotes. Rather than look at each of these features in detail we look at the use of back quotes, used in more sophisticated customization, and aliases, which are a more common use of customization.

A string enclosed in back quotes is executed as a command with the resulting output replacing the back quoted string. To use back quotes, a user generally has a higher level of sophistication. We found that 89% of all users have none or one back quoted string in a shell dot file. Since there is one back quoted string in the default files, it is apparent that the use of back quotes is not a common form of customization.

The system data collection program found that 81% of the users had between one and six aliases. The default shell dot files contained two aliases. The maximum number of aliases was 155 with an average number of six.

The data collected from the volunteer users allowed us to identify specific customizations done by users. From the 224 volunteer users we compiled a list of 4174 aliases. As we see the average number of aliases, and the level of customization, was higher for the volunteer users. We were able to automatically identify a number of categories of aliases.

- 114 of the aliases used the *telnet* command, which allows remote logins to other machines. In particular, many of these aliases were for connecting to interactive games available at other sites.
- 397 of the aliases were used for sending messages to other users. These one-line messages are sent directly to a logged in user's terminal screen.
- 61 aliases were for *finger*ing people across the Internet. This command retrieves information about a user.
- 580 aliases were set up to use locally created programs that were not in normally accessible public areas.
- 447 aliases were used to rename UNIX commands into the equivalent DOS command name, such as *del* as an alias for *rm* to delete a file.

Customization Level of Users

To examine the customization level of users, we divided the users into three groups depending on their use of dot files. A new user begins with only the two default files shown in Table 1. Users who have not modified these default files are grouped together as beginning users. The second group of users are those that have one or more of the other application default files shown in Table 1. Because the creation of these files requires a higher level of sophistication, these users are labeled as intermediate. The last group of users are those that have modified their default shell files and have one or more of the other application dot files. These users have the highest level of sophistication.

Figure 1 shows the number of users at each customization level after discarding users who never have logged into the system. This figure shows that 79% of users have modified at least one of their default files and 22% have also created one or more additional application dot files. These data indicate that most users are incorporating some amount of customization into their computing environment.

We were interested in determining whether all of the beginning users were actually at the same level of customization. Having identified colors as an important first step to customization, we examined the colors for beginning users by looking at the .Xdefaults file. This file manages colors and window set-up. It is interesting to note that of the 572 beginning users, only 13% have no .Xdefaults file, while 87% beginning users do have a .Xdefaults file. We used this information to divide the beginning

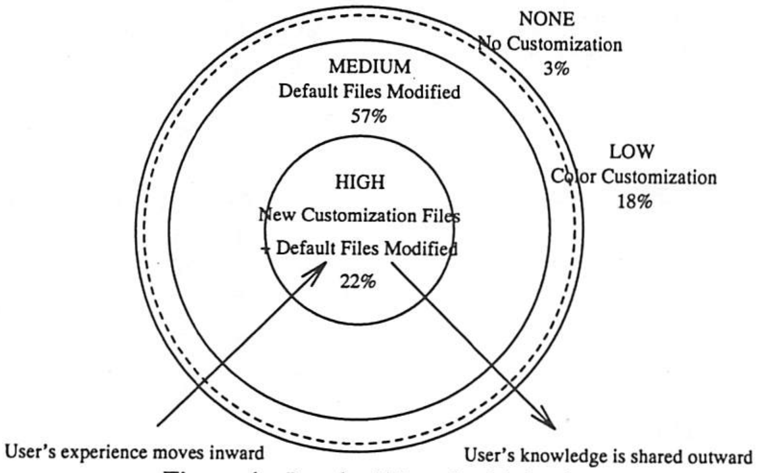

Figure 1: Level of User Sophistication

users into two groups: no and low customization levels. Low users have at least taken the first steps in the process of customization.

From our findings concerning the level of customization on the WPI system, we conclude that almost all users employ some level of customization. Only 3% of the users have done no customization of their computing environment. On the other hand we only have between 15 and 25% of the users with higher levels of customization. Thus, while users appear open to using customization there is a gap between the customization experience of core users shown in Figure 1 and the dissemination of this knowledge to users on the periphery. In addition, the level of customization does not indicate where the customizations came from. For example, some customization comes from invoking programs, such as the session manager or a network news application, that automatically create dot files for the user. Other customization comes from copying customization files of friends. To find answers concerning the origin of customizations users were contacted personally for follow-up interviews.

Interview Results

The volunteer data collection program solicited users for follow-up interviews with a large number of the respondents expressing an interest. To determine which users to interview, a rough measure of the level of customization was computed for each user based on the number and size of dot files, use of constructs such as back quotes and lack of system default files. Thirteen users with a high level of customization were identified. These ''experts'' were interviewed to determine specific customization techniques they employ that could be shared with others and to determine the triggers and barriers to customization, similar to the study by Mackay.

The interviews consisted of three types of questions: specific information about customizations that experts use; ideas suitable for sharing with other users; and information about the culture of customization at WPI.

Findings About Specific Customizations

Specific information was gathered about shell prompts, mailers, use of the *emacs* editor, what commands the user executes when logging into a terminal or workstation, and the most and least used customizations. In terms of mailers most expert users used UNIX *mail*, but *elm* and *emacs* were also used for handling mail.

Our expert users presented an enormous amount of customization for the personalized making the customizations harder to directly share with other users.

It was interesting to explore the use of the DEC session manager, which allows the user to customize certain aspects of the window environment. Only three of the users currently used the session manager for setting their environment, although seven of the experts admitted using it ''in the beginning.'' Three of these users specifically mentioned that they used the session manager to set the colors on their workstation display as their first act as new users.

One example of a specific customization that we explored in detail with our expert users was their choice of a shell prompt. The results are shown in Table 3. As shown, there is no clear preference for a specific prompt, but a number of users had a nonsense word for their prompt (for example ''Zug zug'' or ''woogie'').

history	3
partial directory	3
full directory	3
user name	2
host name	4
"DEC"	3
"WPI"	2
nonsense word	6

Table 3: Prompt Contents

Findings About Culture

The interviews were particularly useful for finding out about the culture of customization on campus – triggers and barriers to its use and how it is shared among users. Table 4 shows that 12 of our 13 expert users customized their system slowly over a period of time as they reacted to needs or worked in bursts. Only one user said that he did all of his customization in one day. As the interview progressed, the user volunteered that he had copied all of his customization from another of our expert users.

Slowly over time	12
All at once	1
Added as needed	6
In bursts	3

Table 4: Customization Timing

The two most common ways of acquiring customization, according to our expert users, were to personally write their own customization and to modify customization techniques given to them as shown in Table 5. Almost all of the users acquired some number of customizations from other users, which compares favorably to the results obtained by Mackay [4]. However, the proportion of expert users writing their own customizations compares to Mackay's findings for system programmers in a corporate environment indicating that high customization users are playing a similar role in a campus environment.

From these interviews we were able to determine that time is a barrier for many users. Almost all of the expert users informed us that their customization was created slowly over a period of time, and most of their customizations were personally written. Before a user is able to implement personally written customization techniques, he or she must have an understanding of the technique. Another barrier which many users commented on throughout the interviews was that the man pages (the on-line system manual) were not written for beginners, so they were hard to learn from.

The expert users also identified triggers to the customization process. Many mentioned that workstation colors and bitmap backgrounds made the environment more appealing to work with, which increased the amount of time and energy they put into customizing the system. Many expert users also mentioned hanging around the computer center and absorbing the culture there as a trigger to their customization process.

Ctool

Our findings show that users are open to customization, but we need a better means to disseminate what and how it can be done. Users are willing to share, but often the sharing is limited to a small subset of friends and in the large, unstructured computing environment the sharing often takes the same form. Sharing entire customization files provides useful customizations for the recipient, but can also leave the user bewildered with many customizations that are not understood.

As a result of our findings *ctool* was developed to help beginning and intermediate users become familiar and comfortable with the computer environment. A key design feature of *ctool* was extensibility so that new customization modules can easily be added by more sophisticated users to aid novice users.

The tool allows users to select customizations from a tree-structured menu, which causes specific lines to be added to the appropriate dot file. When the program adds lines to a file, it also adds comments informing the user that these lines were added by the customization tool, and what the lines do. In this way, the user can begin to understand what happens in specific customization files and begin to experiment with the command options available. At this time, the tool does not replace or delete any customizations previously added by itself or the user.

Initial feedback on the tool has encouraged us that such a tool has a role to play in the community. Even though the customizations are not sophisticated they are useful for novice users to feel more comfortable and therefore in more control of the system. Providing users a means to select individual customization features allows them to understand how their environment is being changed. Because of its extensibility, the tool naturally allows others to add customization modules.

Future Directions

Our work suggests a number of directions for future research. Of immediate interest to us is obtaining more measurable feedback on the effect of tools to facilitate customization sharing. Although *ctool* was a good initial attempt at sharing customizations, it is relatively unsophisticated and needs to be upgraded in how it works and its interface. The operations available in the modules and the ability to recognize what the user already has for customizations would increase the power of the tool. The interface could be made more sophisticated than the initial command-line, menu-oriented one.

Another important question is how to measure effectiveness of customization in a computing environment. A place to start would be a study of tool usage over an extended period, although our experience and others' suggests that measuring the feeling of comfort and control is the real goal. Related to usage of the tool is whether real handymen need to also be available to encourage customization. While such people can be used in cohesive project groups, their effectiveness in a large and diverse community such as a campus appears to be less.

An interesting aspect for future work is to examine the customization habits of users over time. Plotting customization level versus time would help in understanding triggers and barriers by identifying times that customization did and did not occur. The use of tools such as *ctool* should break down barriers and increase the level of customization in a shorter time.

	All	Most	Some	Few	None	Then Modified
Personally Wrote	4	2	0	2	0	
Group Wrote	0	0	1	0	0	
Public Programs / Found	2	0	0	0	0	
Given to Them	3	1	1	1	0	4

Table 5: How Customizations Acquired

Such time/customization plots could also be used to examine characteristics of customizable applications or environments. For example, applications that show a low usage of customization capabilities over extended time indicate that customization is not needed or may be too difficult to use. Application plots that show immediate customization and then a leveling out by all users of the application indicate that the default settings for the application may be incorrect, forcing users to immediately customize.

Summary

In this work we examined the use of customization in a campus UNIX computing environment. This study provides information for customization on a large scale as compared to previous studies, which were centered around users on a specific project. As expected we found a variety of customization use in the system, but were interested to note that almost all users, who had logged in, showed evidence of customization. The least sophisticated users had at least used a session manager tool to customize colors on their workstation indicating an openness to customization. In terms of type of customizations, many users had a number of aliases for shortcuts to interaction with each other. The use of customization for aiding interaction is also seen with the number of users having .friends and .enemies files.

On the other hand, most users still showed a relatively low level of customization, indicating a lack of time or understanding for customizing their environment. We did find the existence of a computing culture where handymen disseminate customizations to other users, but in a large environment we find many users who are not well connected. This situation leaves a gap between the core system users who use customization facilities and periphery users who show an interest in customization, but are unaware of how to pursue it.

These results point to the need for better means for connecting novice users with useful customizations. Towards this aim we explored the introduction of a customization tool, which allows novice users to tap into the customization knowledge of others. Initial reaction of users suggests such a tool can be useful, but it needs more sophistication and more customization modules for continued use.

References

[1] S. R. Bourne. UNIX time-sharing system: The UNIX shell. The Bell System Technical Journal, 57(6):1971-1990, July-August 1978.

[2] Heikki Halme and Juha Heinanen. GNU emacs as a dynamically extensible programming environment. Software-Practice and Experience, 18(10):999-1009, October 1988.

[3] Richard Hesketh. Perly-UNIX with buttons. Software-Practice and Experience, 21(11):1165-1187, November 1991.

[4] Wendy E. Mackay. Patterns of sharing customizable software. In CSCW 90 Proceedings, pages 209-221, October 1990.

[5] Wendy E. Mackay. Triggers and barriers to customizing software. In ACM CHI'91 Proceedings, pages 153-160, April 1991.

[6] Allan MacLean, Kathleen Carter, Lennart Lovstrand, and Thomas Moran. User-tailorable systems: Pressing the issues with buttons. In ACM CHI'90 Proceedings, pages 153-160, April 1990.

[7] Donald F. Norman. The trouble with UNIX. Datamation, pages 139-150, November 1981.

[8] John K. Ousterhout. An X11 toolkit based on the Tcl language. In Proceedings of the Winter USENIX Conference, pages 105-115. USENIX Association, January 1991.

[9] D. M. Ritchie and K. Thompson. The UNIX time-sharing system. Communications of the ACM, 17(7):365-375, July 1974.

[10] Robert W. Scheifler and Jim Gettys. The X Window System. ACM Transactions on Graphics, 5(2):79-109, April 1986.

[11] Richard M. Stallman. EMACS: The extensible, customizable self-documenting display editor. SIGPLAN Notices, 16(6):147-156, June 1981.

[12] Craig E. Wills, Kirstin Cadwell, and William Marrs. Sharing customization in a campus computing environment. In HCI International '93, August 1993.

Author Information

Craig Wills has been an assistant professor at WPI since 1990. Previously he had worked at AT&T Bell Labs. His interests are distributed systems, networking and user interfaces. Reach him via U.S. mail at Computer Science Department, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609. Reach him electronically at cew@cs.wpi.edu.

Kirstin Cadwell graduated from WPI in 1992. She works at Digital Equipment Corporation, where she is an SNMP software engineer for Network Access Servers. She can be reached via U.S. mail at Digital Equipment Corporation, 550 King Street LKG1-3/A12, Littleton, MA 01460 or electronically at kirstin.cadwell@lkg.mts.dec.com.

William Marrs graduated from WPI in 1991. He is at Atria Software Inc. where he works on Configuration Management tools. Reach him via U.S. mail at Atria Software Inc., 24 Prime Park Way, Natick, MA 01760. Reach him electronically at bill@atria.com.

Local Disk Depot – Customizing the Software Environment

Walter C. Wong – Carnegie Mellon University

ABSTRACT

The **depot** model, developed at Carnegie Mellon University, provides a method for managing third-party and locally developed software. **depot** uses an object-oriented approach to managing software; each software package is managed as one or more logical objects. Yet, from the perspective of a user, the multiple ''software objects'' appear as a single, integrated, software environment.

Local disk depot (**ldd**) is an extension to the **depot** framework. **ldd** facilitates the management of environments that ''inherit'' software from the ''master'' software environments. The inherited software environment is formed by taking software and configuration information from the master software environments, and integrating that with local software and configuration information. The most common use of **ldd** is to have an inherited environment on the local disk of a workstation. This allows the workstation administrator to locally cache software in order to improve performance and availability of critical software in the event of server or network failure.

The inherited environments, however, can be used for more than saving local copies of remote software. Workstation administrators can introduce customizations to the software environment, as well as add additional software, even from other software environments. Developers can easily test new or updated applications on their own machines in an environment that is otherwise identical to the released environment.

Introduction

The software release management model currently in use by the Andrew system at Carnegie Mellon University is designed around a tool called **depot**[Coly92B]. In the **depot** model, each component of a software environment is separated into logically managed software objects called *collections*. An example of a software environment would be `/usr/local`, and a sample collection would be all the files and directories that make up the application **gnu-emacs**.

There is a separate directory for each collection. Each of these directories contain the files and directories that make up the collection. The organization of these files and directories reflect how the application should look when exported to the environment. For example, the `info` files for **gnu-emacs** would be stored in the `info` directory of the **gnu-emacs** collection. Similarly, the binary for **gnu-emacs** would be in the `bin` directory, the library files in `lib`, etc.

The separation of software objects is primarily for administrative purposes. The users do not see separate pieces of software, rather they see the union of all the individual software objects, the software environment. Thus, even though the binary for **gnu-emacs** is located in a separate directory from the **X11** collection, the users would still access the **gnu-emacs** binary in the environment's `bin` directory, such as, `/usr/local/bin`, along with the **X11** executables, and all the other executables in the environment.

Up until very recently, the software environments supported by the Andrew system were exported to client workstations via AFS, a distributed file system [Saty85]. The `/usr/local` path was just a symbolic link to the AFS path where the software was stored and managed. The most obvious benefit of this scheme was central maintenance of the software environments. The administrative burden of software installation, customization, and maintenance was removed from the local workstation owner. Local hard drive space was expensive and it was more cost effective to buy larger hard drives and place them on the servers. Finally, there wasn't that much software available for UNIX systems.

Today, the administrative issue is still a strong argument for central maintenance of software. However, disk space is cheap, and there are also many compelling reasons to move software to a local workstation's disk:

- *Local is faster than remote*. Even with AFS, a caching distributed file system, file access is much faster locally than from the distributed file system, even if the application is already cached [Stol92]. Additionally, the fileserver and network are shared resources so your workstation is competing with all the other workstations using those resources.

- *Local has greater availability*. In the case where the applications resides on the local workstation, the only fault that can prevent software from being accessible is a failure on the local machine (assuming there are no license server or other similar dependencies). However, when using a distributed file system, not only will a local problem stop you from getting to the software, you are now also susceptible to network problems as well as any problems that may occur on the servers you are using.
- *Local allows customization*. Software that is centrally maintained limits the options the local workstation owner has in customizing and configuring it to suit his needs. This does not have to be the case, but as the user to administrator ratio keeps increasing, there is even less time that an administrator can allocate to do something specifically for one user or one machine.

depot, by itself, has the ability to create a software environment on the local disk, based on a software environment on a distributed file system. What **depot** lacks the ability to merge local customizations with those from the central services. This is where local disk **depot (ldd)** comes in. **ldd** allows central maintenance of the software environment to co-exist with local configurations of the software environment. In this manner, new software can be introduced to the system, current software updated, and old software removed by the central services without direct intervention from the workstation administrator. Furthermore, any configuration options set by the local administrator will remain in effect throughout any of the central changes.

Implementation

The implementation of local disk **depot**, comprises of three parts. The first is a preprocessor. The second is the directory layout, or tree structure, of the software environment. The final component is a simple shell script wrapper that provides a simple interface to updating the environment.

Rather than making **depot** a "kitchen sink" type tool, we have another program to perform the preprocessing tasks. The preprocessor, **dpp**, takes the local environment configuration file, `custom.depot.proto`, runs it through a macro preprocessor, **mpp**, which expands all the variables and incorporates all of the specified library files. Then, **dpp** merges the configuration information from all the given environments produces and produces `custom.depot`, the configuration file used by **depot**. For example, a minimal `custom.depot.proto` for the `/usr/local` environment, would contain the following lines:

```
%include /afs/andrew.cmu.edu/wsadmin/depot/src/depot.include
*.searchpath: ${local}
```

The `%include` directive is an **mpp** command which incorporates the centrally maintained customization information, including the definitions of various variables. For example, **`${local}`** gets translated to the proper AFS path for each supported architecture. On a DECstation using the default released software, **`${local}`** expands to the path `/afs/andrew.cmu.edu/pmax_ul4/local/depot`. The `*.searchpath` line is normally used by **depot** to locate the software collections that will be integrated into the environment. **dpp** also uses this option to locate and include configuration information. In the event that multiple searchpath elements are given, the `custom.depot.proto` has precedence for configuration information, and then the precedence depends on the order in which the searchpaths are listed. The earlier on the `searchpath` list, the higher the precedence.

The second component is the tree structure. The collections that are released for general use are located under `${local}`, When a new software application is released, the old version is not removed from the environment. Rather the new version is placed in `${local}` with a higher **depot** version number.[1] Our choice for an ever-increasing version number is the UNIX time value, that is, seconds since 1970. For example, for in the case of the `splus` collection, under `${local}` we have:

```
splus.714684343  splus.745275952
splus.746318854  splus.747006152
```

Thus, `splus.747006152` contains the files that are actually available in the environment.

Every 30 days, the environment is examined to see what collections may be removed. The 30 days is an arbitrary limit within which we expect that a **ldd** update will occur on every workstation. At this 30 day interval, any version that is older than 30 days is removed, unless it is the only version out there.

This organization is necessary because files on the local disk of the client workstation may be linked to files that exist on AFS. In order to ensure that these symbolic links continue to point to valid files, one either has to update every client workstation whenever the software environment changes, or one needs to provide a way for the local environment to stay consistent until an update can occur. Updating all the workstations at once did not seem to be a scalable or reliable solution, so the tree organization described above was developed. This mechanism also provides the local workstation owner the ability to set the update time and

[1]When **depot** is run, if uses the collection with the highest version number.

frequency to best suit his needs, as long as it is within the 30 day interval.

The third component is a front end. It has two primary functions. The first is logging and notification. It centrally records the configuration file so that the central facilities have some idea of what machines are out there and what their configuration is. Additionally, this provides a backup of the configuration information. The front end also notifies both the local administrator and the central administrator in the event that an error occurred during an update. The second purpose of the front end is to simplify the interface to the **ldd** system. For example, the front end replaces the need for the workstation administrator to know all the command line arguments that have to be passed to both **dpp** and **depot**. With this interface, all the user needs to do is type in

```
/etc/dodepot  <environment>
```

where `<environment>` is the path to the environment that should be updated, for instance `/usr/local`.

Usage

Copying Files

The most common usage of **ldd** is to move applications that normally reside on the distributed filesystem to the local drive of workstations, and ensure that the applications do not get out of date. A good deal of flexibility is provided for the workstation administrator. He can specify that files, directories or entire collections are to be copied. Alternatively, entire directory hierarchies can be marked as 'link only' in order to conserve local disk space. For example, the workstation owner can specify that all the files (and directories) in `bin` and `lib` are to be copied while all the files in `include` and `man` are to be linked. Thus the workstation administrator can have locally exactly what he wants locally. When any of the software is updated or changed, new versions are automatically copied over.

Testing

ldd has proven to a convenient mechanism for testing applications before they are released to the environment. Developers can incorporate a new collection to be tested by just adding the following line to the `custom.depot.proto` in the appropriate environment:

```
app.path:${destroot}/${sys}/local/app/004
```

In this case, the files for the application `app` would come from that path listed above, even if that collection already exists in any of the directories specified by the searchpath. When the developer is done testing, he can remove the line, run **ldd** and the software environment would be restored to the previous state.

Local Customization

With **ldd** it is possible to install departmental or machine specific customizations to the software environment. For example, we compile **X11** as it is distributed from the X Consortium. Any local changes are made in the collection called `x11config`. The files in `x11config` overrides, or replaces, the appropriate files in `x11r4`. If individual workstation owners or departmental workstation managers wanted to have a different `x11config` collection, they could simply use **ldd** to specify a different path to `x11config`, and thereby tailor **X11** without having to compile it themselves.

Another customization example would be the installation of software packages that aren't normally available in the central software environment. The primary benefit of installing software in the same location as the central software is the users would access the new software just as if they were access software provided by the central services. No paths or other configuration information would have to be changed in the user's home directory.

The only danger in customizing the software environment is that the users may get confused. Before **ldd** was available, all Andrew workstations shared the same software environment (e.g. `/usr/local`, and `/usr/contributed`). With **ldd** users may see different defaults and different software depending on if they are using a ''departmental'' Andrew workstation or a ''public'' Andrew workstation. So far, this has not been a problem, neither do we foresee it to be one.

Software Sharing

Software sharing under the **depot** model is very straightforward due to the organization of the software components. In the event a distributed file system is not available, then one could simply **tar** up the collection, transfer it to the remote machine, and then extract it. If the remote machine did not run **depot**, one could just extract the collection in the appropriate directory (such as `/usr/local`).

A more sophisticated example would be to use **ldd** to enhance the departmental software environment. Some departments on campus wish to minimize the dependency on the central services. For example, if the central fileservers or the network to the central services went down, they want to still be able to get their work done. As a result, packages that are considered ''critical'' are compiled are exported via NFS. AFS is still used but only for ''non-critical'' applications, as such, `/usr/local` is a symbolic link to the central software repository and another directory is allocated for the mission critical software. With **ldd** the software environment could still be exported via NFS, if so desired, but the mission critical software could be copied to the local disk of the NFS fileserver, using the methods described previously. This potentially saves the departmental administrator a good deal of time compiling and installing software that has already been compiled and installed. Software that isn't provided by the central services, could, again, be installed locally.

Mobile Computing

ldd provides a simple way of supporting mobile computing and disconnected operations via the copying mechanism previously described. Software used when disconnected can be copied to the local disk and updated each time the workstation is connected via a high bandwidth network, such as Ethernet. At this time, all the linked software is also available, as usual. When the workstation is disconnected, then only the copied applications are available. When the workstation is connected via a low bandwidth network, like SLIP, then the other applications are still available, but access would be considerably slower.

The component nature of **ldd** also facilities mobile computing by making a simple GUI possible. For example, there could be a 'point and click' options for what to copy and what to link as well as buttons for 'connect' and 'disconnect.'

Problems

The first problem is that there is a reasonable amount of wasted resources between **dpp** and **depot**. Both these programs contain code that perform virtually the same functions. This is being addressed with the second major version of **depot**. The new version of **depot** provides a programming interface, so any action that **depot** performs can be accessed internally from another program. Work on rewriting **dpp** to use these libraries is about to commence.

As with any system that automatically updates software, there exists the problem of updating applications that are currently running. As with many other systems, we ignore the issue. Traditionally, this problem is addressed by doing the update when the machine reboots. In general, we have been able to ignore this issue since it has not been a problem.

The use of version numbers to maintain local disk consistency potentially wastes a significant amount of disk space, especially when you increase the update window. The only way around this is to force all the workstations to update when the central environment is updated. Given those choices, we decided to sacrifice the disk space.

Finally, version numbers tend to increase the complexity of the system and give the administrator more work to do. In small sites, it may not be worth using the version numbers for this reason and thereby sacrifice a degree of consistency. For larger sites, this may not be acceptable and what we have is a set of support tools that handle much of the work. An overview of these tools can be found in [Coly92A].

Future Directions

The work on the next major release of **depot** is nearing completion. The two major changes in this release of **depot** include significant performance enhancements and a programming interface. As discussed previously, **dpp** will be rewritten to use these programming libraries, especially since the file format for the configuration file has changed in the new version of **depot**.

The most interesting work is the use of the preprocessor **mpp**. **ldd** takes a step forward in doing workstation administration via libraries and via a more modular/object oriented approach to managing all the files on a workstation. For example, the local administrator will be able to issue specific commands to enable (or disable) specific functionality, such as, a single command to enable anonymous FTP, or a command to disable **fingerd** from running. The end goal is to let the local workstation owner manage as much of the software as he wants in a simple and effective manner, and let the departmental and/or central management services fill in the gap in such a way to maximize the local, departmental, and central resources.

Conclusion

Originally, **ldd** was designed only to move applications off of AFS and thus speed up day to day work. However, as work progressed we saw that the benefits were not limited to just that. It has proven to be a a simple and easy method for testing, customizing, and sharing software. **ldd** has been in use at our site for over a year now, and we foresee it as tool to help us in our mobile computing plans as well as in our general workstation management strategy.

Availability

dpp is available from export.acs.cmu.edu (128.2.35.69) in the /pub/depot directory. **dpp** is currently in use on HP/UX 9.x, Ultrix/RISC 4.2A, SunOS 4.1.3 platforms. The code is in ANSI C.

An internet mailing list for **depot** and **ldd** exists. To subscribe, send email to info-cmu-depot-request@andrew.cmu.edu.

References

[Coly92A] Colyer, Wallace; Held, Mark; Markley, David, and Wong, Walter. "Software Management in the Andrew System." AFS *User's Group Proceedings.* Spring 1992.

[Coly92B] Colyer, Wallace, and Wong, Walter. "Depot: A Tool for Managing Software Environments." *LISA VI Proceedings* 1992. pp. 153-162.

[Held92] Held, Mark, and Neuhart, Dawn. *Software Management in the Andrew Distributed* UNIX *System at CMU.* Computing Services, Carnegie Mellon University. 1992.

[Saty85] Satyanarayanan, M.; Howard, J. H; Nichols, D. A.; Sidebotham N., and Spector A. Z. "The ITC Distributed File System: Principles and Design." *Proceedings of the 10th ACM Symposium on Operating System Principles.* 1985.

[Stol92] Stolarchuk, Michael T. "Faster AFS" AFS *User's Group Proceedings.* Spring 1992.

Author Information

Walter Wong graduated from Carnegie Mellon University with a B.S. in Cognitive Science in 1991 and promptly joined the Andrew Systems Group performing various programming and system administration tasks. His current focus is on workstation and software administration issues. Feel free to contact him electronically at wcw+@cmu.edu. Alternatively, U.S. Mail can be addressed to: Computing Services; Carnegie Mellon University; 5000 Forbes Avenue; Pittsburgh, PA 15213-3890.

Appendix A: Sample custom.depot.proto

The following is an "interesting" **custom.depot.proto** from a DECstation 5000 client running AFS.

```
%define beta
# This changes the ${local} variable to point to the
# 'beta' software tree (e.g.
# /afs/andrew.cmu.edu/system/beta/pmax_ul4/local/depot
%include /afs/andrew.cmu.edu/wsadmin/depot/src/depot.include

*.searchpath:/afs/andrew.cmu.edu/usr13/ww0r/devel/depot/local, ${local}
usemodtimes: true
# use the modification time to determine whether or not a copied file
# should be updated.

# create a variable for my convenience
%define destlocal ${destroot}/${sys}/local


# Software currently being tested. Use the software for these
# collections located at the given path, rather than the files
# for the collections found in the searchpath
depot.path:${destlocal}/depot/020
dpp.path: ${destlocal}/dpp/021

# symlink these files/directories under /usr/local
linktarget:root.client,root.server,man,lib/X11/XP
linktarget:bin/sas,lib/sas/sas

# copy these all the files/directories under these directories
copytarget: bin/rlogin,bin/rsh,fonts/andrew,lib/X11
copytarget: lib/sas/sasexe/base,lib/sas/sasexe/graph,lib/sas/sasexe/stat
copytarget: bin/lpr

# copy all of these collections
wcw.mapcommand: copy
graphon.mapcommand: copy
afs.mapcommand: copy
frame.mapcommand: copy
gnu-emacs.mapcommand: copy
x11r4.mapcommand: copy
x11config.mapcommand: copy
perl.mapcommand: copy
telnet.mapcommand:copy
rlogin.mapcommand:copy
tools.mapcommand:copy

# ignore these directories/files under /usr/local
specialfile: lost+found,tmp
```

Methods for Maintaining One Source Tree in a Heterogeneous Environment

Bjorn Satdeva – /sys/admin, inc.

ABSTRACT

A common problem when maintains software in a heterogeneous environment are the need to maintain executable for several platforms and/or operating system versions. Ideally this should be done from a single source tree. This papers gives a overview of The *Depot, Xheir* and the *make*(1) as it appears in BSD Net2.

Further, the BSD Net2 make is compared to The Depot and Xheir, and some practical experiences with the BSD Net2 make are be presented, and its advantages and shortcomings are outlined.

Introduction

A common problem when maintaining software for a heterogeneous environment are the need to maintain executable for several platforms and/or operating system versions. Ideally this should be done from a single source tree.

Two solutions which partially addresses this problem, *Xheir* and *The Depot* have been presented at previous LISA conferences. However, a third solution, the new BSD *make*(1) first distributed in the BSD 4.3 Tahoe, and now part of the BSD Net 2 release, has received little attention in the LISA community. The BSD Net2 make (hereafter referred to as the Net2 make), has, in my opinion, several very significant advantages over both Xheir and The Depot. The Net2 make do not require use of the automounter, and the symbolic links required (one per directory) are all build and maintained by Net2 make itself. Further, Net2 make knows which platform it is building binaries for, and provides include files and an extended command set, allowing most make files to be only a few lines long.

Although Xheir and The Depot have as their main focus the distribution of compiled software, and they place less emphasis on the development and maintenance issues, it is the only previous work I have found addressing these issues (the *imake* from the X11 distribution addresses maintenance of multiple platforms, but cannot do this simultaneously). This paper gives first a brief overview of source maintenance aspects of Xheir and The Depot, followed by a more complete description of the Net2 make. Finally, some practical examples will be presented of the work which where required for a few software packages in order to use Net2 make to build them, and the tools which can be an aid when maintain the ported software.

Overview of The Depot

The Depot is primary a methodology for distributing software. However, as it provides some for compiling sources for multiple platforms, it has been included here. It uses an elaborate scheme of mount-points and symbolic links to appear as only a single directory tree for a given platform is mounted. It has a limited support for single set of platform independent sources, but there is no evidence that compiles on multiple platforms can occur simultaneously.

The Deport release very heavily on NFS mounts, loopback mounts, and symbolic links, as well as NIS and the SunOS automounter. It appears as a very complex strategy, which attempts to solve simultaneously the handling sources, distributing of executables, and handling of mounting of the binaries for the correct platform. In spite of this, it appears to leave many aspects of a single source tree unresolved.

Overview of Xheir

Xheir is both a development environment and a distribution tool, but as The th Depot, the main emphasis is on the distribution. Like The Depot, it makes the source tree available on different platforms, however, it mainly does so by distribution, rather than sharing through NFS.

The Xheir package has a vary large number of commands (80) and provides a more automated approach to the distribution issue than The Depot. However, like The Depot, it does not address the issues of maintaining a single source tree very well.

Like The Depot, the Xheir package places a high usage on symbolic links, but it does not rely on neither NIS nor the automounter. Like The Depot, the many symbolic links makes the package complex and more difficult to understand. The authors claim that the use of the extended command set enables the maintainers of the various packages to perform their work without understanding the underlying structure. However, it is still required that *somebody* understands it well enough to fix problems when something is broken.

Overview of BSD make

BSD Net2 make is backwards compatible with the previous versions of make. However, it has been redesigned to provide a new higher usability, not found in the previous versions. It differs from The Depot and Xheir in the respect that it is aimed at maintaining a single source tree for multiple platforms while using a single source tree. All object files are kept outside the source tree, enabling the source tree to be NFS mounted read-only on the build machines, if so desired. It is therefore not necessary to jump through hoops to keep the object files for the various platforms when compiling from one source tree. In addition, the Net2 make knows which platform it is running on. This information is kept in the pre-defined variable *${MACHINE}* which can be used either to specify machine type to the compiler through a *-D${MACHINE}* or by using a conditional statement (on this or other variables) at execution time.

To create a directory to keep the object files separate, one needs to type the command *make obj* which will create a directory for the object files (by default below the directory /usr/obj). A symbolic link from the source directory (named *obj*) to the object directory, is built by the Net2 make by typing the command *make obj*. Before starting a compile, make ensures that all files created during the make will therefore be placed in the obj directory. If it is necessary to make explicit references to files in the source directory, it can be done by using the variable *${.CURDIR}*

In a traditional makefile, dependencies must be listed explicit, although some versions of UNIX can create the list of dependencies by way of *makedepend*(1) and insert this information at the end of the makefile. For Net2 make this is no longer necessary, as the command *''make depend''* will call *makedepend*(1) and placed the result in the file *.depend* in the object directory. This not only removes the need for the bad hack of rewriting the makefile, it also allows for each platform keeping its own dependencies, independent of any other platform.

In addition, much of the content we expect in a traditional makefile has been moved into generalized make libraries or include files which are made available by the *.include* command.

However, all this nice functionality comes at a price. In order to to take full advantage of the functionality of BSD Net2 make each program must be located in its own private directory. It is therefore necessary not only to rewrite all makefiles but also to move some, if not all, source files into a new directory structure. However, for most programs, this is only a relative minor task, which can often be done in less than one hour. Overall, the advantages therefore, in my opinion, far outweighs the disadvantages.

An simple example of a Net2 makefile for a single program, is the makefile for the *ps*(1) program. As *ps* consist of only one executable and one man page, it do not require the sub–directories mentioned above. The Net2 makefile for ps is shown in Figure 1.

```
PROG=    ps
SRCS=    devname.c fmt.c keyword.c
SRCS+=   nlist.c print.c ps.c
CFLAGS+=-I/sys
DPADD=   ${LIBMATH} ${LIBKVM} ${LIBUTIL}
LDADD=   -lm -lkvm -lutil
BINGRP= kmem
BINMODE=2555
BINDIR= /bin

.include <bsd.prog.mk>
```

Figure 1: Complete BSD Net2 *Makefile* for ps(1)

In spite of the absence of any visible targets in this Net2 make, there is nevertheless everything necessary to compile all the object modules, link the executable, format the man-page, and install the executable and man page in the correct directories.

Comparisons of The Three Methods

Whether to use The Depot, Xheir, or Net2 make will naturally depend on local practice and preferences. However, I personal much prefer Net2 make over any of the other solutions. The primary reason is that both The Depot and Xheir intermix software development and support with software distribution; two very different problem sets, which best are handled as separate issues. In addition, both appear to be complex to setup and operate. By using Net2 make for the support and development, it is possible to maintain a single source tree for a large number of platforms, with only two requirements: support of symbolic links and a working port of Net2 make No other programs are necessary, no support for the automounter is required, and there is no large system to support.

Because Net2 make does not have the support for distribution, another means must be provided. At all sites where this has been an issues, *Fdist,* has already been in use. This package handles the distribution aspect of software maintenance very well. The use of Fdist for distribution purposed, also provides a solution for packages approach provided in Xheir. However, this has required writing both a include file for Net2 make, and make some accommodations in Fdist previously missing. The solutions provided for this is still work-in-progress, and the entries supporting this have been removed from the Net2 makefiles in the appendices.

One item which curiously appears to be absent in both Xheir and The Depot are the ability to install the software on the build machine, and test it before distributing to wider use. Not all software packages lend themselves well to testing directly in the development location (it might relay on parts of the executables and/or configuration files be found in specific locations). By actually installing the software package and running it on the build machine, better regression testing can be provided, reducing the risk of distributing a faulty software packages. The combination of Net2 make and Fdist allows the person in charge of a given software packages, to build and test the package, and then explicit releasing it to the user community, whenever, the testing has been completed satisfactory.

Xheir provides a facility for adding necessary information to files such as */etc/services* when a new application is added. This is possible a necessary feature at a very large site. However, having system files altered, and possible damaged by automatic installation programs are in my opinion not a desirable solution. With Fdist in place, it is a relatively minor job to modify files such as /etc/services on a site wide basis. Therefore, in the environments described in this paper, no substitute for automatic altering of system files, as provided by Xheir are provided.

The attempt of comparing The Deport and Xheir with Net2 make is a little bit like comparing apples and oranges. However, there is a sufficiently amount of overlap of the problem set that the three tools attempt to solve. In the realm of maintaining software for different platforms, the Net2 make outperforms both Xheir and The Depot by far. With the support of additional tools, such as Fdist and Modules, I find that the sum of maintaining the sources, and make the resulting programs available to the users are less than for either it would be with either The Depot or Xheir.

Porting Sources to BSD Make

As explained above, it is, in addition to provide new make files, necessary to create a new directory structure, and move the various source files into the correct sub-directories. The first few times I did this, the process where somewhat time consuming. However, as the strategy becomes clear, this can be done for many programs in less than hour. Large software packages, such as *c-news* or *smail*3.2 will naturally take a good deal longer.

It has already been shown what the make file for a single executable will look like. However, most packages consist of at least a few different executables. Below is two examples of such conversions. In addition, I will discuss the conversion of *c-news* as an example of a large package.

As a general strategy, I identify each executable, and create a directory for each. I then move all "dot-c" files which is only part of that program into the newly created sub-directory. All include files is moved into a single sub-directory, names "1include", and any leftover "dot-c" files are moved into a directory named "lib". I then build first the library, and the each of the individual executables. When all parts can be build without any errors, they can be installed by the command *make install* and the newly build programs can then be tested on the build machine, before distributed to other users.

Net2 Makefiles for Less

Less where the first package I ever converted. The less package comprises two programs *less* and *lesskey* each with their own man page. Less comes with a configuration script, *linstall*, which builds two platform dependent files, *defines.h* and a *makefile*. A complete and correct port should, in my opinion, include support for this script under Net2 make but this being the first port I did, this was never done. In this case, the *linstall* command was instead moved into a separate directory, *etc*, and the *defines.h* file built as necessary. This file is then moved to another directory, named *include.${MACHINE}*, and included in the c-compilers include search path. This is a hack, but it works sufficiently well.

There is a couple of interesting issues illustrated by this makefile. More than one line is necessary to define all the source files for *less*. The += operator concatenates the definitions, so there are no need to escape the newlines. Because this is a local program, it is necessary to override of the default for where the man page shall be installed. This is done by explicitly setting the *MANDIR* variable. Because *less* needs an additional text file with all the help information, an *afterinstall:* target has been added. This will ensure the help file is installed after the executable. The complete makefiles are shown in Appendix A.

Net Makefiles for nntp

The port of nntp (which was done after c-news) took about twenty minutes to complete. The approach where the same as outlined above, but was supported by nntp already having some support for the "one program, one directory" strategy. The support for "make client" and "make server" was maintained.

Some issues which help illustrate the use of Net2 make are the globally included makefile *Makefile.inc*. This file imports the value of *${NEWSBIN}* from cOws. Also, the two common files, *version.c* and *clientlib.c* are not placed in a library, but are instead found by make, using the

.PATH: directive. The makefiles for nntp are shown in Appendix B.

Experience Porting C-news

Porting c-news to the Net2 Make where a much bigger project, and not only because of the size of package. C-news is split into several sub-directories, with a similar, but not identical, structure as used for the binaries. In addition, it used a number of shell scripts which has to be executed in sequence, by a number of different users (''root'', ''bin'', and ''usenet'). These shell scripts are in turn build by yet another shell script, which builds a number of configuration files.

I have always found this process cumbersome and unattractive, both because of its unusual style and because it do not lend itself well to a large source directory tree, which can be build from the top. I therefore decided, when I had to rewrite the makefiles and the build shell script, I would do it in a manner which I found better suited for my use.

As the result, the build script is rewritten to build just a configuration file for subs (a shell script which is part of the c-news distribution, and a very useful tool in it's own right). All other files which must be created to fit the local customization all build from make. This approach gives the entire make process an interesting twist, as the many of the variables set and used by make are set in a included makefile, built by make. To ensure support for multiple platforms, it was further necessary to place one of the included makefile in the obj directory structure,

The two major problems encountered in doing this port were ensuring all command line defines for the c-compiler were included correctly in the new make files and ensuring that all files where installed in the right place. The first issue where all resolved through the extensive regression tests in c-news, and the latter through trial and error.

Effectively, the time to do this port was about one week. However, much time where also spend on porting the c-news system to the BSDI platform, and to deal with problems created by bugs in the early version of *sh*(1)

The update from the c-news Performance release (using Update, discussed below), to the Performance Plus release took less than one day, including the time to inspect all files which differed between the two releases. This indicates that with appropriate tools, continued maintenance of a software package (once ported to the Net2 make) does not include a inappropriate overhead, but rather, that the time and effort saved by truly having one source tree, is well worth the effort using the Net2 make.

Maintaining Ported Sources

From the very start, it was clear that some support tool was needed for applying updates to software packages, which where distributed with traditional makefiles, but which was locally ported to the Net2 make. There are really two different kinds of problems which needed to be addressed. Updates of minor nature can be handled by Larry Wall's *patch* program; however, once in a while, a program has so many changes that it is resubmitted as complete source rather than as a series of patches.

The support of *patch* had already been resolved by Jeff Polk from BSDI, who readily made his script available. However, the *c-news* package is always released in full, even when the changes are relatively minor. There were two possible avenues to take: either the current port could be discarded and the new release be reported (not a very attractive option) or write a utility which could perform the necessary update by comparing the official released package with the locally supported version, and update those files which differed, The latter approach was chosen, resulting in a perl script currently named *Update* in lieu of a better name.

This tool compares the contents of two directory trees, and attempts, based on file names, to decide where files from the first directory should be moved into the second, in order to make update. The problem with this approach, is that some files in a large distribution, such as *c-news* will have identical file names (think just at files such as Net2 make and *README).* In *Update*, this has been resolved by creating a small table to resolve such conflicts. *Update* first finds all files in both source and destination directory and then starts resolve how files in the source directory should be mapped to files in the destination directory:

1. Resolve mapping of files which full source and destination pathnames are specified in the *Update* mapping file.
2. Resolve mapping of all files whose full path is identical both directory tree's.
3. Map all file names (basename) which are unique in both source and destination directory.
4. Resolve mapping of files which have added an extension in the destination directory (as for example a *Makefile* in the original distribution is renamed to *Makefile.org* in the destination directory
5. Resolve mappings which can be done by specifying a source directory and a destination directory.
6. Repeat mapping of all remaining files whose names now are unique in both directory trees (some ambiguity has probably been removed by now).
7. Acknowledge all ignored files.

Update does not copy any files directly. Instead, it generates a combined report and shell script which includes the necessary commands to copy files which content which differ from the source to the destination directory. This makes it possible to run *Update* as often as necessary, inspect the result, and perform the actual copying only when the output has been deemed correct.

Update was written as a fast hack in less than one day. The result is as could be expected. The commands in the *Update* mapping file are obscure at best. The available commands are shown in Figure 2, the complete map file for c-news are shown in Appendix C.

b	map by base name
D	map source dir to destination dir
e	map by basename plus extension
P	map by path name
i	ignore file
w	ignore path,
I	ignore file, but display warning
W	ignore path, but display warning

Figure 2: Codes used by *Update*

However, the principles used in *Update* have proven themselves very useful. The program has proven to be able to resolve a correct mapping of the 522 files in the c-news distribution, with only 46 explicit mapping instruction in the mapping file. This provides with reasonable assurance that a new version of *c-news* (or any other similar large package), can be ported to our platforms in a reasonable time.

Porting BSD Make to Another System

Net2 make is only useful if it can be used on all system, so sources must be supported. It is therefore necessary, as a prerequisite, to port it to all such systems. The sources for Net2 make comes supposedly with a makefile for this. However, that makefile is completely useless, because a key definition is absent. As part of porting Net2 make itself, it is necessary to decide on a string describing the platform (such as sun3-40 for a Sun3 running SunOS 4.0, or sun4-43, for a Sparc running SunOS 4.3). This name should be defined at compile time in the define MACHINE. In addition, one or more of the functions from the Net 2 distribution which are not available on the target machine (such as *findenv.c, setenv.c,* or *strerror.c*) may need to be copied from the Net 2 sources, and included in the source tree for Net2 make. Appendix D shows parts of a traditional make file (all lines showing the object dependent on the source has been removed to preserve space). This makefile assumes that additional functions necessary to compile the programs on a new platform are in the directory *${MACHINE}*.

Distribution

As mentioned earlier, Net2 make has no means to distribute files; that distribution is a task to be performed separately from software development. By using using a dedicated distribution system, such as *fdist,* to provide the necessary distribution, this can be kept separate from the development system. *Fdist,* as a dedicated distribution system, using a domain oriented addressing scheme, gives a much greater freedom in specifying where files shall be distributed to, without complicating the development environment. From the distribution point of view, the package concept, as seen in *Xheir* are implemented through use of *Fdist* domains, and from the user's perspective, through use of *Modules.*

However, at this time, Net2 make and *Fdist* are not very well integrated, making it a bit frustrating to update the distribution copies of a file. Work is currently in progress to integrate *fdist* and Net2 make. The goal is to enable the developer to type *make distribute* in order to update the currently distributed files.

Conclusion

In my opinion, Net2 make in conjunction with other packages, such as *Fdist* and *Modules,* provides much of the same functionality as found in *The*Depot and *Xheir* with less complexity and fewer requirements to the underlying operating system. As Net2 make is a general software building tool, in the same manner as the tradition *make* program, it also has fewer assumptions about the site, and requires less work to port to new platforms. One could hope, that, as BSD Net2 and BSD 4.4 get more widespread usage, more developers on the net would start to provide some support for Net2 make even if initially, only to support the one program, one directory concept. For people who are required to maintain sources for multiple platforms, this would be an easier world to work in.

Availability

The Net2 Make is part of the BSD Net2 lite distribution, and is available from ftp sites carrying the BSD Net2 sources.

Update where posted to alt.sources beginning of this year. It can also be obtained by sending a request to bjorn@sysadmin.com.

Author Information

Bjorn Satdeva is the President of /sys/admin, inc., a consulting firm which specializes in Large Installation System Administration. Bjorn is also President and founder of Bay-LISA, a San Francisco Bay Area user's group for system administrators of large sites, and Columnist for SysAdmin, the UNIX System Administration Magazine. Contact Bjorn via US mail at /sys/admin, inc.; 2787 Moorpark Avenue;

San Jose, CA 95128; by telephone 408 241-3111, or via e-mail at bjorn@sysadmin.com

References

Manheimnen, K., B. A. Warshaw, S. N. Clark, and W. Rowe, "The Depot: A Framework for Sharing Software Installation Across Organizational and UNIX Platform Boundaries", LISA IV Proceedings, October 1990, pp 37-46.

Wallace Colyer &and Walter Wong, "The Depot: A Tool for Managing Software Environments" LISA VI Proceedings, October 1992, pp 153-159.

John Sellens, "Software Maintenance in a Campus Environment: The Xheir Approach" LISA V Proceedings, October 1991, pp 21-28.

Bjorn Satdeva & Paul Moriarty, "Fdist: A Domain Based, File Distribution System for a Heterogeneous Environment" LISA V Proceedings, October 1991, pp 109-128.

John L. Furlani, "Modules: Providing a Flexible User Environment" LISA V Proceedings, October 1991, pp 141-149.

Make(1) manual page for BSD Net2

bsd.README file from /usr/src/share/mk directory

Appendix A: BSD Net2 Makefiles for *less*

Makefile:

```
SUBDIR =           less lesskey .include <bsd.subdir.mk>
```

less/Makefile

```
PROG=    less
SRCS=    brac.c ch.c charset.c cmdbuf.c command.c decode.c edit.c filename.c
SRCS+=   forwback.c help.c ifile.c input.c jump.c line.c linenum.c lsystem.c
SRCS+=   main.c mark.c optfunc.c option.c opttbl.c os.c output.c position.c
SRCS+=   prompt.c screen.c search.c signal.c tags.c ttyin.c version.c

BINDIR= /usr/local/bin
MANDIR= /usr/local/man/cat
LIBDIR= /usr/local/lib
HELP=    less.hlp

CFLAGS+=-I- -I${.CURDIR}/../include -I${.CURDIR}/../include.${MACHINE}

DPADD+= ${LIBTERM}
LDADD+= -ltermlib

funcs.h:         ${.CURDIR}/../include.${MACHINE}/funcs.h
${.CURDIR}/../include.${MACHINE}/funcs.h:        ${SRCS}
        cd ${.CURDIR}; awk -f mkfuncs.awk ${SRCS} > \
                                            ../include.${MACHINE}/funcs.c

afterinstall:
        install -c -o ${BINOWN} -g ${BINOWN} -m 555 ${HELP} ${LIBDIR}

.include <bsd.prog.mk>
```

lesskey/Makefile

```
PROG=            lesskey
SRCS=            lesskey.c

BINDIR=          /usr/local/bin
MANDIR=          /usr/local/man/cat
CFLAGS+=-I- -I${.CURDIR}/../include -I${.CURDIR}/../include.${MACHINE}

.include <bsd.prog.mk>
```

Appendix B: BSD Net2 Makefiles for *Nntp*

Makefile

```
SUBDIR= doc inews

.ifnmake install
```

```
SUBDIR+= mkgrdates server xmit xfer shlock
.endif

install_server:
        cd server; make install
        cd xmit;   make install
        cd xfer;   make install

.include <bsd.subdir.mk>
```

inews/Makefile

```
.PATH:  ${.CURDIR}/../common ${.CURDIR}/../server
NEWSUSR=usenet
PROG=   inews
SRCS=   inews.c uname.c postauth.c clientlib.c version.c strcasecmp.c
NOMAN=
BINDIR= /usr/local/lib/nntp

install:
        install -c -o ${BINOWN} -g ${BINOWN} -m ${BINMOD} ${PROG} ${BINDIR}

.include <bsd.prog.mk>
```

mkgrdates/Makefile

```
PROG=   mkgrdates
SRCS=   mkgrdates.c
BINDIR= ${NEWSBIN}

.include <bsd.prog.mk>
```

server/Makefile

```
.PROG=   nntpd
SRCS=   main.c serve.c access.c access_inet.c access_dnet.c active.c
SRCS+=  ahbs.c globals.c group.c help.c ihave.c list.c misc.c netaux.c
SRCS+=  newgroups.c newnews.c nextlast.c ngmatch.c post.c parsit.c scandir.c
SRCS+=  slave.c spawn.c strcasecmp.c subnet.c time.c xhdr.c fakesyslog.c
SRCS+=  batch.c auth.c timer.c version.c
BINDIR= /usr/libexec

.include <bsd.prog.mk>
```

xfer/Makefile

```
PROG=    nntpxfer
SRCS=   nntpxfer.c get_tcp_conn.c fakesyslog.c
NOMAN=

.include <bsd.prog.mk>
```

xmit/Makefile

```
PROG=   nntpxmit
SRCS=   nntpxmit.c remote.c llist.c get_tcp_conn.c xmitauth.c
SRCS+=  fakesyslog.c strcasecmp.c

.include <bsd.prog.mk>
```

Appendix C: *Update* mapping file for c-news

```
#       Source                  Destination                     Comment

# Mapping using file name extensions
e       -                       .SH
e       -                       .PROTO

# Files ignored, based on basename
i       -                       Makefile
i       -                       Makefile.inc
```

```
i       -                           README.bsdi
i       -                           CHANGELOG.bsdi
#
g       -                           Makefile
g       -                           Makefile.inc
g       -                           README.bsdi
# Mapping using basename
b       Makefile                    Makefile.org
b       makefile                    Makefile.org
b       build                       build.org
# The update files
I       inject/rnews                -                         # empty file
I       dbz/dbz.h                   -                         # identical to h/dbz.h
I       -                           conf/build                # The new build command
# Directory mapping
D       h                           hfiles/h
D       dbz                         dbz/dbz
D       libbig                      libs/libbig
D       libbsd42                    libs/libbsd42
D       libc                        libs/libc
D       libfake                     libs/libfake
D       libcnews                    libs/libcnews
D       libsmall                    libs/libsmall
D       libstdio                    libs/libstdio
D       libusg                      libs/libusg
D       libv7                       libs/libv7
D       libv8                       libs/libv8
D       libusg                      libs/libusg
D       doc                         man/doc
D       man                         man/man
# Duplicate file names which need some guide in resolving
P       explode/morefds.c           explode/explode/morefds.c
P       relay/morefds.c             relay/relaynews/morefds.c
P       explode/trbatch.c           explode/explode/trbatch.c
P       relay/trbatch.c             relay/relaynews/trbatch.c
P       relay/active.c              relay/relaynews/active.c
P       rna/active.c                rna/readnews/active.c
# Special moves,
P       misc/README                 maint/README
# Files which are part of makefiles in original distribution
W       -       aux/proto/active.PROTO          Extracted from Makefile.org
W       -       expire/proto/explist.proto      Extracted from expire/Makefile.org
W       -       expire/regress/regress          Extracted from expire/Makefile.org
W       -       dbz/regress/regress             Extracted from dbz/Makefile.org
```

Appendix D – Sample traditional *Makefile* for building Net2 Make

```
MAKE =  /usr/bin/make
OBJS =  arch.o buf.o compat.o dir.o hash.o job.o main.o make.o\
        str.o suff.o targ.o cond.o parse.o var.o

MACHINE=sun4-41
CFLAGS = -O -I. -DMACHINE=\"${MACHINE}\"

all:    machine objs
        cd ${MACHINE}; cc -I. -c *.c
        cd lst.lib; cc -I .. -DMACHINE=\"${MACHINE}\" -c *.c
        cc *.o lst.lib/*.o ${MACHINE}/*.o -o make

clean:  machine
        cd ${MACHINE}; rm -f *.o
        cd lst.lib; rm -f *.o
        rm -f *.o

objs:   machine ${OBJS}

machine: FRC
        if [ "${MACHINE}" = "" ] ; then  exit 1 ; fi
        if [ ! -d ${MACHINE} ] ; then  exit 1 ; fi

FRC:
```

Towards a POSIX Standard for Software Administration

Barrie Archer – ICL

ABSTRACT

The POSIX draft standard for Software Administration is about to go to ballot for acceptance as a formal POSIX standard. Since this standard is likely to form the basis of future Software Administration products it will have a profound effect on the facilities available to administrators and the way they manage software. This paper explains how the standard came about, gives a summary of the features and explains how systems administrations can, via the balloting process, have an influence on the final standard.

Introduction

The distribution, installation and control of software is an important and time consuming task for administrators. Most vendors supply tools for their own systems, but, especially in a network of heterogeneous systems, administrators have often had to resort to inventing their own methods. Previous papers at LISA have reported on some of these efforts. To address this problem the POSIX Systems Administration Group (P1003.7) set up a subgroup to propose a standard for software administration. Working from the specifications of existing tools this subgroup produced a draft standard that will shortly be balloted for acceptance as a POSIX standard.

The purpose of this paper is to bring to the notice of a wide audience the impending ballot of the draft standard and to encourage participation in the ballot as well as to explain what is in the draft standard and why. In describing the draft standard more emphasis is put on the overall structure and the background to what is there, than expounding the detail. By doing this it is hoped that reviewers will appreciate the conflicts that were addressed and the process that led to their resolution. They will then be in a better position to understand what the draft standard is trying to achieve and will be able to contribute to maintaining a coherent standard.

This paper was prepared whilst the draft standard was still under development and so anything stated here should be taken as a guide only - refer to the standard itself for definitive information. Also, for the sake of readability, some simpler terms have been used in this description in place of the formally defined terms in the draft standard.

Objectives

The subgroup defined three objectives that the standard should address.

Administrator Portability

By providing an interface for software administration that was consistent on all conformant implementations, administrators would be able to use any such system without retraining.

Standard Packaging format

A common packaging format would enable software to be processed on any conformant system. This does *not* imply a architecture independent format, although it does not preclude it. Software can only be run on an architecture it is designed for. It does allow, however, for discless clients to be catered for.

Distributed Administration

The provision of interoperability interfaces enables distributed software administration across systems. This can be done either through the command line interface or through a management application specifically written for the purpose.

Standards

In order to be useful, a standard must define interfaces or formats in a sufficiently rigorous way that there should be no ambiguities that could result in incompatibilities between implementations. How this applies to POSIX standards is discussed in a later section. However, attaining this necessary rigour does not lead to a readable document. For example, any particular aspect should only be defined once in a standard, whereas for readability a summary of the aspect might appear in several places.

Rationale

In order to try to address the problem of readability POSIX standards contain sections of *rationale*. These sections are intended to explain what parts of the standard mean, how they are expected to be used and why they are there. Even the addition of rationale has its limitations, however, and cannot substitute for the kind of overview being presented here.

Scope

Another important consideration in defining a standard is to limit its scope to something that can be achieved in a reasonable time. There is a trade off here between what one would like to do and the least one can do for a usable standard. In the section on the history of this standard there are some comments on how the scope changed over the definition life cycle. An aim of this paper is to give some information about how the standard came about and why it covers some things but not others. It is hoped that this will enable those who join the balloting group to be in a better position to make comments.

POSIX Standards

POSIX Standards have to be approved by the Project Monitoring Committee who will seek to assure that the standard is reasonable, that it is based on existing practice and that there is sufficient support to enable the work to be done. Once such approval is obtained a group (or sub-group as in the case of Software Administration) can be formed which will meet at the quarterly POSIX meetings to progress the development of the draft standard. At the end of the development process a draft will be produced which will be balloted

Balloting

To ballot a draft standard a balloting group is formed. The IEEE uses appropriate means to advertise that the group is being formed. Any individual may join, but comments from those who are not members of IEEE or the Computer Society are for information only. To pass the ballot 75% of the balloting group must respond and 75% of those responding must agree to the standard. Agreement can be the result of comments being taken into account - the process known as ballot resolution. Of, course if there are too many comments requiring material changes it would be necessary to ballot again.

Mock Ballot

It is customary for groups to engage in a *mock ballot* prior to the ballot described above. The intention is to address the same audience and to find out if there are any fundamental problems before going to ballot. For Software Administration the mock ballot showed that Configuration, Recovery and Software Service (patching) would have to be addressed in the draft to go to ballot.

ISO Standards

Once standards have been approved through the balloting process they go forward to ISO for ratification as international standards. This is handled by Working Group 15 of SC22. There are certain agreements in place between IEEE and ISO designed to smooth this process by ensuring that the POSIX standards will be acceptable to ISO without alteration. One area where this affects the work is that a POSIX standard can only reference other formal standards. It cannot reference or rely on an implementation or de facto standard.

History

This section covers the way in which the draft standard evolved. This is useful information for understanding why the draft standard contains what it does and why the facilities are defined in the way they are.

Participation

One of the conditions of starting out on the process of producing a POSIX standard is that there should be sufficient commitment to enable the work to be done. The subgroup was fortunate that there were was a high level of commitment by several companies and individuals. Most major vendors were represented as were users, in the form of living/breathing systems administrators. The subgroup was also able to get work done between meeting by the use of a mail reflector. In this way even those who were not able to attend a particular meeting could continue to contribute.

Existing Practice

Another condition for a POSIX standard is that it should be based on existing practice and not be a invention of the group, hence indicating that the standard can be implemented. One of the first actions taken by the subgroup, therefore, was to examine the existing practice, and this was done by inviting submissions, either or both of a paper submission or a presentation. The companies that made such submissions are given in Table 1.

IBM
ICL
Digital Equipment Corporation
Hewlett Packard
SNI
SCO
UNIX Systems Laboratories

Table 1:: Companies making submissions

The subgroup found that all the submissions had many features in common and that there was a good deal of agreement in the facilities that should be provided. Obviously, some features were only found in some submissions and there were some misalignments. Nevertheless, the subgroup was encouraged by this to proceed, in the belief that there was a good chance that the interested parties could come to an agreement on a draft standard.

It should be noted that the requirement for existing practice brings its own complications. These can arise because existing practices in different areas being addressed by the draft standard do not fit together well, or because there is no one existing practice that gives all the facilities identified as necessary for the draft standard.

Comparison

The subgroup drew up comparisons of the documents submitted in order to determine the core facilities and to examine the additional aspects of particular submissions. Part of this work appeared in the rationale of the draft that was basis of the Mock Ballot.

Base Document

In order to start work on the text of the draft standard, the subgroup decided to adopt one of the submitted papers as a base document. The one chosen was the SDU utilities submitted by Hewlett Packard (also the basis of OSF DME software distribution). Having adopted this base document the group then proceeded to modify it so that it was more generic and also covered important features not found in the SDU utilities but which existed in other submissions or identified as needed by the mock ballot.

Mock Ballot

The document that was distributed for the Mock Ballot was Draft 8. It was recognised that a lot of work needed to be done on it before it could become a formal standard but it was felt that the time was appropriate to get a wider opinion of the work so far. 67 people took part in the Mock Ballot, sending in almost 1000 comments. Three of these responses were classed as votes against the proposed draft standard, the rest being qualified approvals. Many respondents identified the lack of rigour, but there were also many comments that pointed out problems that might not otherwise have been corrected before the formal ballot. In addition it became very clear that Configuration, Recovery and Software Service (patching) would have to be addressed to make the draft standard acceptable. In getting to draft 8 these items had been considered and dropped due to a lack of existing practice and in an attempt to simplify the task of producing the draft standard. What the Mock Ballot clearly showed was that many people saw them as vital parts of the standard.

Overview

This section gives a high level description of the draft standard, the details of which are filled out in later sections.

Components

The draft standard can be considered as consisting of three key components, which are required to achieve the objectives.

Packaging Layout

The draft standard defines the information that is held about software on a distribution medium, as well as the way this information is represented on the medium. This definition enables the use of different media to distribute software (including electronic transfer), optimising the use of each type of media according to its particular attributes. The draft standard does *not* define an architectural neutral format but does not preclude it. However, it does allow for the architecture of a product to be identified and for variants of the product for several architectures to be present simultaneously. Hence, an appropriate variant may be chosen from several on a medium.

Commands

The draft standard defines commands for performing the various tasks that are needed in order to perform software administration. The definition of these commands is based on the submissions received. On any conformant system an administrator will hence have a consistent way way of dealing with software.

Management Information

The draft standard defines the information which describes the software being managed. The draft standard does not define how this information is stored for software that has been installed, although it does define the way in which the tasks use the information. The management information is sometimes referred to as the Management Information Base (MIB) by analogy with Network Management standards, although this term will not appear in the draft standard.

Roles

In order to provide a framework for producing the draft standard the concept of roles was used, although it is not in the normative part of the draft standard (although it is in the rationale). The concept of roles helps in the explanation of the tasks but it is not rigorously defined and so could not be included in the normative part of the draft standard. This is just one example where the rigour of a draft standard conflicts with making it readable. Figure 1, shows the relationship of these roles, which are further discussed below. Note that this diagram is a simplified version of the one that appears in the rationale of the draft standard. The different roles may each take place on separate systems or combinations of roles may take place on one system.

Package Role

In the package role *developed software* is taken and put in a *distribution.* How the developed software got into a state to be packaged is outside the scope of the draft standard - the standard is not intended to cover the area of software development and source control.

Source Role

In the source role a distribution, or one or more components of it, is transferred to where it is to be installed. The transfer may also be to another source role - a staging operation. This transfer may take the form of electronic transmission or transfer on some medium. The concept of the source role came from some of the submissions which had very extensive functionality associated with this area. However, other implementations provided much less functionality and allowed for the role to effectively null in some circumstances.

Target Role

It is in the target role that the software is installed, that is it is deployed and manipulated to put it in a form that will enable it, eventually, to be run. One important aspect of the target role is that it may take place on a system which has a different architecture from that on which the software will run. Where there are discless clients the target role is taken by the server.

Client Role

In the client role the software is configured so that it will be in a state to be run. Configuration takes place on the architecture on which the software is to run. Installed software may be subject to being configured several times, for example comms software may be configured to serve several different paths. At mock ballot configuration was outside the scope of the standard because it was believed to be in the scope of another subgroup. However, many responses to the the mock ballot indicated that without configuration the draft standard would be incomplete and of significantly less use. Since the other subgroup had not progressed their work, configuration has been added.

Manager Role

Having the manager role provides for distributed control of the software administration process. The functions performed in the other roles can be controlled by the manager role. The manager role may be performed by the command line interface provided in the draft standard or by a management application. It is worth stressing again that the manager role can be on the same system as any (or all) of the other roles.

Developer Role

This role is specifically outside the scope of the standard. In the developer role software is *constructed* and placed into the form known as *developed software* which is the form in which it can be accepted by the package role. Typically this will involve activities such as compilation, source control, etc.

Structure of Packaging

The draft standard defines very precisely the format into which the software is packaged, this being the form that the source role transfers. The draft standard also defines the format for developed software, particularly the steering information which defines how to do the packaging. The draft standard does not define the format for installed or configured software (this being specific to particular architectures). The rest of this section gives the high level structure of the packaging format.

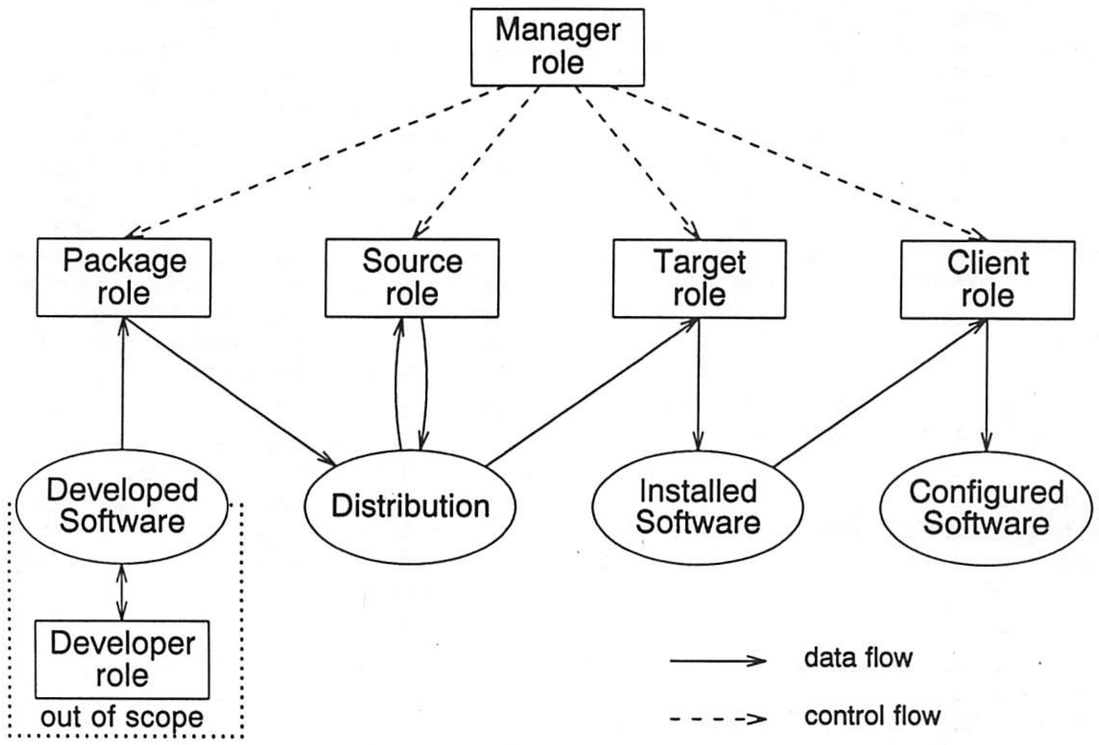

Figure 1: Roles in Software Administration

History

Although the various submissions showed considerable commonality in the fundamental concepts of the packaging format, this area of the draft standard caused significant problems, undergoing substantial changes before and after mock ballot. The problem lay with how many *levels* of structure there should be in the packaging format. *Products* containing *filesets* containing *files* was an obvious and simple format but one which all submitters had found to be inadequate. At one point (the first Santa Clara meeting) a recursive structure was proposed whereby a product could contain a product, and this could be to any depth. However, this was not what is commonly understood by the term "product". The group hence looked for some other (unloaded) word but ended up adopting, temporarily, the term RNC - for Recursive Notational Convenience!

However, although the recursive structure had many attractions, it was an unknown quantity, having no known existing practice. It was also felt that when work progressed to the detail, let alone implementation, there would be significant problems caused by this structure. An example might be dependencies (q.v.).

After many discussions over many meetings, the structure illustrated in Figure 2 has been adopted. This has subproducts within products and bundles within distributions.

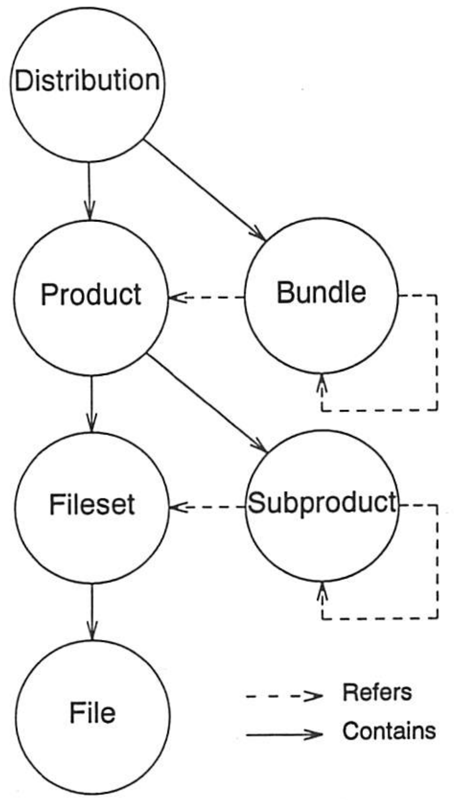

Figure 2: Structure of Packaging Layout

Products

Although products were common to all submissions it took some effort to tightly define what they were since there were significant differences in the detail between the submissions. A late addition to the draft standard is the concept of *bundles* to group together products. These are explained below. Products are defined in the draft standard to have attributes like a name, a revision number, architecture, etc. One area of concern was that two software vendors might produce products with the same name. The draft standard already incorporated a mechanism to permit different versions of software to co-exist in a distribution and in installed software. However, in order for administrators to correctly identify a product, a vendor tag was added as an attribute that could be used to select a product. The group realised that there could still be a conflict if vendors used the same tag but felt it was beyond their remit to solve this problem. However, the administrator can also display the vendor description attribute where a vendor can put additional information, such as address, support telephone number, etc. There is probably little chance of this not being unique!

Products contain filesets and subproducts. Products can have dependencies on other products (see section on Dependencies).

Filesets

A fileset is a collection of files that are logically related. The important point about a fileset is that it is the smallest unit that can be specified for the tasks defined in the draft standard. Filesets have many of the attributes of a product, such as name, version, architecture, etc. Filesets can have dependencies on other filesets, as well as bundles, products and subproducts (see section on Dependencies).

Subproducts

Subproducts are contained in products and are a method of addressing a group of filesets or subproducts. Hence a fileset (or subproduct) may be referred to from more than one subproduct. Subproducts do not contain anything and are not the recursive structure mentioned above. Subproducts are very simple and have few attributes (no revision or architecture, for example). A use of subproducts might be to group together the man pages, thus allowing an administrator to load a product, or products, but not the man pages for them.

Bundles

Bundles enable several products to be grouped and managed together. A major example of this was the operating system, which is a collection of products distributed as a whole. Bundles refer to products or other bundles in a similar manner to subproducts. They share many attributes in common with products. Products exist in a distribution in their own right; they do not have to be referred to from bundles. Some details of the operation of bundles is still being worked out by the group. There are discussions taking place about their attributes and the extent to which they still exist in installed software.

Packaging Information

The packaging format defines two types of information, the data that is the actual software (code, data, resources, etc.) and the control information that enables the installation process to take place. It is this control information that actually supplies the structure discussed in the preceding sections. In order to make installation efficient from a serial medium this information is required to be at the start of such a medium.

Tasks

In this section the tasks that are provided by the draft standard are described. These tasks are invoked using the CLI commands defined in the draft standard or by applications.

Phases

Tasks are implemented in three phases, the selection phase, the analysis phase and the execution phase.

Selection Phase

In the selection phase the filesets that are to be the subject of the task are determined. A fileset may be included because is has been specified individually or because it is part of a higher level component (e.g., a product). The way in which a selection is specified on the command line is covered in a later section. In addition a fileset may be included because it (or a component it is a part of) is needed to satisfy a dependency. In this case the fileset may be included without being specified to the task.

Analysis Phase

The analysis phase determines if the task is likely to succeed. This involves evaluating if there are enough resources, whether dependencies are satisfied, etc. Success in this phase does not guarantee that the task will succeed but failure should only occur if the task would certainly fail. A key aspect of this phase is that no change is made to the system so that if the phase fails part way through no recovery is necessary to revert to the initial state. The analysis phase is run for all selected products and filesets before proceeding to the next phase.

Execution Phase

In the execution phase the actual work of the task takes place, using the information from the selection phase.

Packaging

The task of packaging takes place in the packaging role and involves collecting the components of the software, together with control information, and making this into a distribution. The draft standard defines the way the steering information is supplied to the task as well as the way in which the component files are supplied. The information supplied to this task involves a detailed knowledge of the software and how it is constructed. It is envisaged that this task will be performed by the implementors of the software, either directly or as part of a make(1).

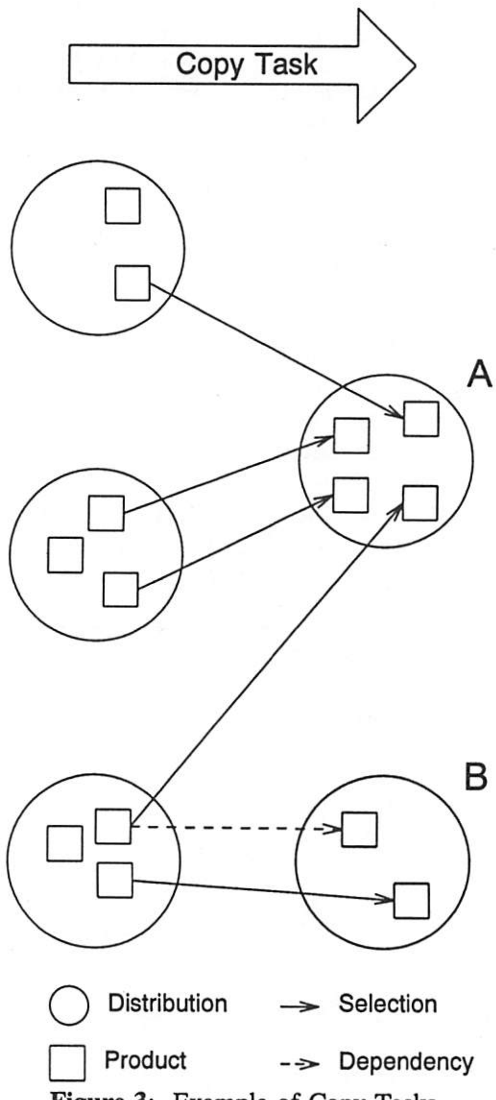

Figure 3: Example of Copy Tasks

Copying

Copying takes place in the source role and involves copying complete distributions or parts of them. Where parts of distributions are to be copied, the selection mechanism is used to define the components to be copied. Where the destination already exists, copying involves adding to the distribution. Copying may take place to or from different storage forms, for example copying to a serial medium.

It is envisaged that copying will be a common task performed by administrators. It might involve taking several distributions, received from the implementors or a software distributor, and constructing one or more further distributions from them. These new distributions may contain only parts of the original distributions, with only some products from bundles being copied or some subproducts from products. The latter case may occur where, for example, it was decided not to distribute the tutorial components of a product.

Figure 3 shows an example of creating two distributions, A and B. Distribution A is created by selecting 4 products from 3 distributions. Distribution B is created by selecting one product from a distribution and a second product is also copied because the selected product has a dependency on it. A similar example could show filesets or subproducts being selected from within products or products from within bundles.

Installing

Installation takes place in the target role and involves transforming software from the distribution format to the installed form in which the software can configured to be run. This involves operations such as creating directories, copying files, setting permissions, running scripts, etc. The installation process is explained in more detail in a later section. Input to the installation process may be a distribution in filestore (possibly copied from a serial medium) or directly from a serial medium.

Configuring

Configuring software is the final step before software is actually made operational and, unlike installation, always takes place in the client role and on the client architecture. The definition of configuration depends on the software but it is expected that it will normally be an operation that can be performed in significantly less time than the installation task. An example might be the installation of a new revision of a Message Transfer Agent. Configuring would specialise the software for the particular situation and make it the revision actually in use. Software may be configured more than once, each giving rise to a different configured instance. Taking the example of the MTA again, it may be that the software is configured for several different services. The parameters to configuration are specific to the software being configured, and are hence not part of the draft standard. They are supplied via the request task.

Removing

Removing a product involves deletion of the filestore elements that were created during the installation process as well as the management information relating to the product. There are some elements that are not removed, these being information that users would wish to have left. Examples of this would be the files that make up a database or the postbox in a Message Transfer Agent. These are identified specifically in the control information when the product is packaged.

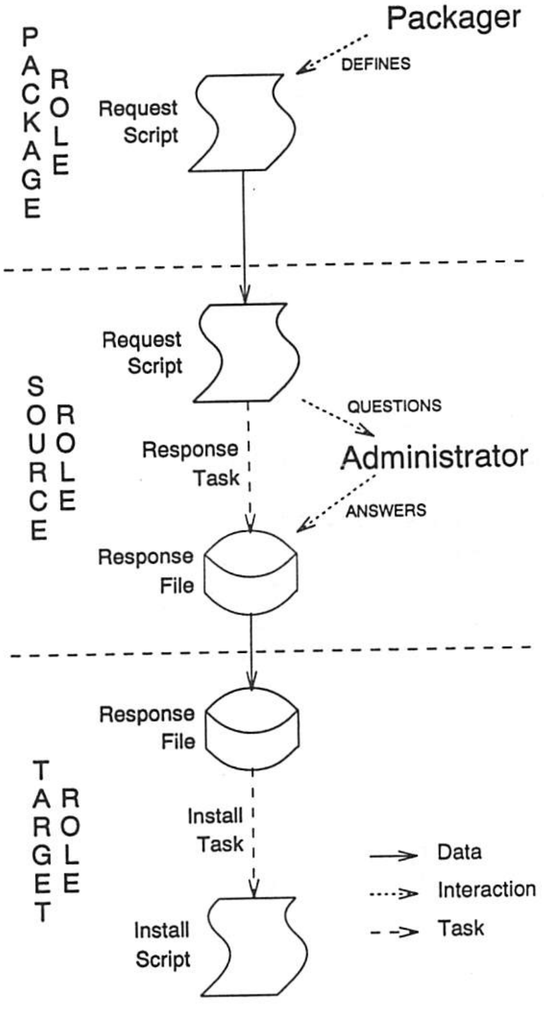

Figure 4: Request Task

Request

The installation and configuration tasks can involve the running of scripts defined during the packaging task. These scripts may need to obtain information to customise the work they do. If the scripts were to interrogate the administrator at the time the information was required, the installation or configuration task would be running interactively. To avoid this undesirable situation a script is defined during the packaging task which asks the questions. This script is run by the request task and the responses stored in a *response file*. When the installation or configuration task is taking place the scripts can use the information from the response file. The request task can be run entirely independently of the installation or configuration task and the response files distributed with the products to which they apply. If this is not done, the request task will be run at the start of the installation or configuration task. Figure 4 illustrates the use of the request script and response file.

Verifying

Verification of a distribution or installed software can be run in the source, target or client roles. It establishes the integrity of the information by checking the file attributes and checksum against the control information. Files that might change, and therefore should not be verified, are marked as such in the control information. If a customisation script (described in a later section) changes the contents of a file, the modification task should be used by the script to ensure that the management information is updated.

Listing

Listing can take place in the source, target or client roles and gives information about distributions and installed software. The selection process determines the items to be listed. The depth of information is given by an option.

Modification

Modification takes place in the source, target or client roles and is the process by which the management information is changed to reflect the information it refers to. This may be necessary because a customisation script has modified a file or because some of the management information associated with a product is inapplicable in a particular situation. Systems administrators who worked on the draft standard emphasised the importance of being able to correct the information, when (not if) it got out of step with reality. Since the way in which the management information is stored is implementation dependent it is necessary to provide a task to change it.

The Installation Process

One of the major items of the draft standard is how the installation of a product (or group of products) takes place. Installation of software involves those activities needed to transform it from the distribution to a state in which it can be run once it is configured.

Files

A fairly straightforward aspect of installation is the creation of directories to hold filesets and the copying of files from the distribution into the installed software. It is also possible to create links. The following sections discusses some of the more complex aspects of installation.

Dependencies

Dependencies provide an important way to ensure that software is correctly installed, configured, copied or removed. During the selection phase of a task a check is made for dependent software. If dependencies are not satisfied the task will fail (this can be overridden). A dependency is an attribute of a fileset that refers to a bundle, product, subproduct or other fileset. A dependency may also be an attribute of a product, which means that it applies to all filesets within the product.

Consideration was given to allowing dependences as attributes of subproducts but this was dropped because subproducts are references not "containers" and the rules would have been too complex.

The following sections describe the three types of dependency defined in the draft standard.

Prerequisites

A prerequisite must already have been installed before the software that depends on it is installed or it must be installed as part of the same installation task. During the selection phase of a task, products may be added to the selected set in order to satisfy prerequisite dependencies.

Corequisites

In the case of a corequisite, the software that is depended on must be installed and configured in order for the dependency to be satisfied. Dependencies such as this might occur for parts of the operating system.

Exrequisites

In this case installation cannot take place if the exrequisite has already been installed or has previously been selected during the install task. Dependencies such as this might occur where versions of a product cannot co-exist on a system.

Customisation Scripts

A common feature of all submissions was the use of scripts to allow installation and configuration to be customised. These scripts are defined during the packaging task. The scripts (apart from the configuration script) are run in the Target Role and thus not necessarily in the environment or on the architecture on which the software will be run. Environment variables for the scripts define the final environment. The method of returning information from scripts is also a problem and a totally satisfactory solution has yet to be found.

Scripts may be associated with products and with filesets. In principle each different fileset in a product could have a different script. Existing practice indicates that such a situation would be unusual and that product scripts are likely to be the most common. A exception to this might be filesets that make up the operating system.

Check Script

The check script is run during the analysis phase of the task to supplement the checks done automatically. For example, the automatic check for sufficient disc space could be supplemented by a disc space check that is dependent on some customisation of the installation specified in the response file. Since the scripts are executed during the analysis phase, they are not allowed to make any modifications to the target role.

Installation scripts

There are two installation scripts, the pre- and post-installation scripts run before and after the files are copied from the distribution. These scripts are run in the environment of the target role, not the client role. Examples of such scripts are the production of a new version of a product by applying changes to a previous revision and the transformation of data into a new format for a new revision of the product. Virtually the only constraint on these scripts is that if they modify the installed software a call to change the management information must be made (modification task). These extensive possibilities raise problems for the draft standard since it is difficult to ensure that the rules given are sufficient to guarantee interoperability. It is probably for this reason that so much discussion within the group concerned this aspect of the draft standard. To enable recovery to take place there are also *undo* scripts for pre- and post-install.

Removal Scripts

Like the installation scripts there are pre- and post-removal scripts. The draft standard does not define what the removal scripts should do except that they should reverse any changes that the installation scripts have made and which have not, or could not, be reflected in the management information. Hence if an installation script creates a file and adds this to the management information such a file will be deleted automatically. However, if a data file needs to be transformed into a different format that will have to be handled by the removal script.

Configuration Script

These scripts perform functions that must take place on the architecture on which the software will run or which are associated with the configuration of a particular instance of the software. Since the only substantive action defined in the draft standard for the configuration task is the running of the configuration script, a product can only effectively be configured if such a script is supplied. The parameters to the configuration task are supplied to the configuration script by means of the request task. Examples of configuration scripts are a compilation to the architecture of the client role or the definition of particular services.

Product Location

The packaging layout specifies a default location in the filestore where a product will be installed. This can be overridden by an option to the task.

Simultaneous Versions

It is possible to install different versions of a product simultaneously, provided the product can be installed anywhere in the filestore hierarchy (i.e., it is relocatable). The version of a product includes its revision and the architecture it is to run on. Hence, it is possible to have simultaneous installation of multiple revisions of a product as well as installing versions for different architectures (important for servers of discless clients). Depending on the product, it may or may not be possible to configure multiple revisions simultaneously.

Overlaying

Only one product can exist at one location. If an attempt is made to install another product (or another version of the same product) at the same location it will either be rejected as an error or the original product will be deleted. The action to be taken can be selected by an option to the task. It is expected that all products will be relocatable and the installation of a new version of a product will not be done by overlaying.

Recovery

Recovery is the process of undoing the effect of a failed task, addressed here in terms of installation but also applying to copying and, to a lesser extent, packaging and configuring. Recovery is only significant when a product has been overlaid. Where a new version is installed simultaneously with an old version, recovery merely involves removing the partially installed new version. As has been stated, recovery was not addressed in the draft that was circulated for mock ballot. This was because the discussions up to that point had not produced a consensus on what should be done. However, responses to the mock ballot showed that recovery would have to be addressed in the final balloting draft. In the event of a failure there are basically two choices, to delete what has already been installed or to leave what has been done so that a subsequent installation does not have to re-install parts already successfully installed. The choice of these could be an installation option. The following sections discuss some of issues involved.

Overlaid Products

Information was provided to the group about implementations that provided recovery by roll-back or by copying and deletion of the old version. Whatever the implementation there are implications in terms of storage required, already a potential problem area if both the distribution and installed software were present on a system.

Administrative Applications

Applications that provide an advanced interface to Software Administration would handle recovery in their own style. In order to enable this to happen the distributed interface would provide detailed control over the phases of the installation process (events on completion of a phase and control over the transition between phases). Any facilities in the draft standard must therefore cover the requirements of the command line as well as administrative applications.

Level of recovery

The components selected for installation may be the result of a high level definition ("install this bundle") or a low level definition ("install these filesets"). It might be deemed necessary for the recovery action to be different in the two cases - and all the cases in between and combinations. However, this seems to imply that the draft standard should contain a very complex definition, detailing what should happen in each case, and providing equally complex overrides for the default actions.

Scripts

When an installation fails it is necessary to run scripts to undo the changes made by the installation script(s). However, it would be difficult for an implementor to ensure that such scripts would work irrespective of the the type or position of the failure.

Current Situation

The current proposal being worked on in the draft standard provides a fairly straightforward recovery mechanism. It is applicable to the situation where a product is overlaid and requires that, in the event of a failure, the product is restored to its original state. Two new scripts are proposed, the unpreinstall and the unpost-install scripts which undo any changes made by the corresponding install scripts.

Interactions During Tasks

One area where there was not commonality in the submissions was the facility for the installation scripts to ask questions of the task submitter. Since making the installation process interactive is undesirable some submissions effectively forbade any such questions whereas others enabled the questions to be answered at the start of the installation task and even allowed the answers to be distributed with the software. The draft standard adopts this latter approach - see the request task.

Software Service

Software Service is the term adopted to describe modifications made to product other than replacement by a different version of the product. This includes replacement of one or more files and in situ modification of the data within a file, the classic form of "patching". When this was initially considered many different methods of achieving it were described. However, there was no common core that could be discerned in these methods and the submitters were frequently not enthusiastic about their own methods. It was therefore difficult for the group to select an existing practice to standardise and for this reason it was omitted entirely from the mock ballot version of the draft standard. The rest of this section describes some of the issues and the current state of the draft standard.

Level Identification

One of the major topics for Software Service was the identification of the modifications that had been applied. In the completely general case each modification would be separately identified and the list of modifications would be available as an attribute of the modified product. However, this does not answer the question of how a task could check that a new modification was appropriate for the existing modification level of a product. Various schemes were in current use, from those that re-issued all previous modifications with each new modification to those that left it up to the administrator to select the modifications to be applied, handling any dependencies or exclusions between them.

Reversion

It is obviously necessary to be able to remove a modification from a product and in the general case this would either require a roll forward from the original, unmodified, instance of the product or would need roll back information to be kept.

Management Information

Modification of part of a product requires that the management information be updated. This would then enable the verify task to operate correctly and not report an error with respect to a modified product. With a roll back provision for reversion (see above) the modified management information would have to form part of the roll back log.

Current State of Standard

The current state of the draft standard is that there will be no additional facilities provided specifically for software service, although the rationale will explain how it can be achieved. This involves the overlaying of one fileset with a new version that has one or more of the files changed. The installation scripts can be used to provide roll back, identification and dependency checking. This is the only solution that seemed capable of accommodating the diverse schemes currently in use.

Installing the Operating System

While the draft standard does not address all aspects of operating system update and initial installation, it does provide the basic functionality so that it can be used as a fundamental part of these processes. Facilities provided include marking files as being part of the operating system and indicating that a product or fileset will not become effective until a re-boot occurs. Excluded, however, are the final stages of switching from the old to the new version of the operating system, which would take place during the configuration stage. The initial installation of the operating system on an empty system requires special techniques since services that are normally assumed to be present (e.g., the filestore) are not available. The draft standard only deals with installation of software when a POSIX compliant operating system is present and so is not applicable to the initial installation of the operating system until this is true. This does not, of course, preclude a vendor from providing such facilities but they would be extensions to the standard.

Tasks using the CLI

One of the objectives for the draft standard has been stated as *Operator Portability*, meaning that, on any system that complies with the draft standard, an administrator would find a well known set of Commands with which to perform software administration tasks. Nevertheless, it was recognised that the interface to software administration, and particularly distributed software administration, would increasingly be the province of an integrated interface, particularly one based on a Graphical User Interface. Such an interface would have a significant advantage where software was being distributed to, and installed on multiple machines simultaneously, a task which is inherently asynchronous. Several of the submissions indicated that such implementations already existed.

The Command Line Format

All submissions provided commands to invoke the tasks and the draft standard was based on these. The basic form of a command is

```
  command [options...] selections \
                       [@ target ...]
```

meaning the the command operates on the software identified by *selections* and the tasks take place on the hosts specified by *target*. The format of the options and target is covered in the rest of this section. The selections, being a significant issue in their own right, are covered in another section. The commands implement the tasks already defined.

Options

An important issue that had to be addressed was the sheer number of options that had to be accommodated. Not all options from all submissions were included but there was an inclination to accept that if a facility had been found necessary or useful it should be included. This issue of the number of options had already been addressed by the sub-group working on the Print standard, P1003.7.1, and a compatible approach was adopted. This involved specifying options in a quoted string given as the -x option to the command, or in a file, the pathname of which is specified in the -X option. Within the quoted string, or file, an option would consist of an identifier and a value. The identifier consists of lower case letters and underscores. These identifiers are not localisable to other languages.

Host Definitions

The format for specifying the machine on which the task is performed is

```
 @ target...
```

and this was generally liked as being intuitive although it does not have any applicable precedence as a separator of operands (its use in mail aliases and Berkeley commands is different). This syntax does not appear in POSIX.2 but is legal according to the utility guidelines of that standard. As distributed utilities extend the problem space that POSIX.2 addresses, avoiding extensions was not deemed to be essential. In the end, the decision of the working group was that the use of @ was acceptable, and indeed desirable over alternatives such as moving the operand to the options.

Selections

A selection defines the items that are the subject of an operation, for example a selection might define the software products that are to be installed from a distribution. At the simplest level this would just be the name of a product. However, there were several areas where the selection got more complex and there was a struggle to achieve the necessary flexibility without a grossly complex syntax. The following sections describe the details of the selection and the objectives that were being addressed.

Depth

A selection can specify bundles, products, subproducts or filesets and so can be as specific or general as required. The implication is always that all the components of the item specified are selected.

Versions

In the draft standard, the version of a product is an attribute that defines its intended architecture, identifies the vendor and provides the revision of the product itself. Hence there can be several versions of the same revision of a product, each for a different combination of hardware and operating system. The specification of the architecture in the selection provides wild cards and the comparison of the revision takes into account the common dot format, e.g., 2.03.

Locations

A selection may also specify the location where the product is to be located as a result of the operation, overriding the default in the product. For example, for the install task the location would define where the product is to be installed. This feature of a selection is a bit of an oddity because the rest of the selection is concerned with the source of the operation whereas the location is concerned with the destination. However, it is necessary because there may be several selections each needing to be located in a different place.

Dependencies

Selections are also used for dependencies, that is for references from a product or fileset to a bundle, product, subproduct or fileset, but in this case the location cannot be specified.

Customisation

The systems administrators who had participated in the development of the draft standard had emphasised the importance of avoiding fixed restrictions whilst at the same time enabling defaults and limits to be set for any particular installation. The existing practice supported this concept and and so this facility was built into the draft standard.

System Wide Defaults

On any system there will be one defaults file which gives the defaults for about 25 aspects of software administration. In addition different defaults can be specified for different tasks. So, for example, the default for whether to try to automatically resolve dependencies could be set to **true** for installation but **false** for copying a distribution.

Local Defaults

The system wide defaults can be over-ridden by the options file to a particular command, a file which is in a similar format to the system wide defaults file. These in their turn can be over-ridden by what is specified on the command line.

The Software Catalogue

The term catalogue applies to a distribution or to a collection of installed software. Most of what the draft standard defines about a catalogue is the control information, which is actually very similar between the two. The difference is that the format of a distribution is defined by the draft standard whereas the format for the catalog for installed software is not. In this latter case it is implementation defined how the catalog is stored although the draft standard does define standard ways of accessing it. For example, the list task reads it and the modify task changes it.

Contents of a Catalogue

Information in the catalogue defines the contents (bundle, product, etc.) of a distribution or installed software, giving the attributes of the components (name, revision and dependencies for example).

Multiple Catalogues

A valuable contribution from Systems Administrators in the group was the need for multiple catalogues, for example corresponding to development software, software under test and production software. The draft standard hence allows for the catalogue to be specified as part of the syntax of the commands. This does however raise the question of how one might be able to find all the catalogues on a particular system. It would be a distinct advantage if this could be achieved in some way but so far this has not been incorporated in the draft standard.

Filestore Structure

The draft standard is based on a POSIX compliant filestore structure but does not specify any other detail about how installed software should be mapped other than that there must be a node under which the product files are installed. This requirement does not exclude the possibility of some files being located elsewhere although this is discouraged.

Software Layout

Software should be constructed so that it can be installed relative to any point in the filestore hierarchy. This is particularly important for the simultaneous installation of multiple versions of the same product. However there are some types of software for which this is not possible, particularly the operating system itself. In such circumstances the software will have to be constructed to provide some other method of handling simultaneous versions, possibly by some special action as part of configuration.

Alternative Root

Sometimes it is necessary to install software relative to a virtual (or alternative) root. This means that absolute references in the installation to the filestore hierarchy are taken to be relative to the alternative root. This is particularly useful for installing operating system software for a discless client or on a disc unit that will be installed in another system (preloaded software). The discless client example is illustrated in Figure 5 in which software is installed on the target with node D as the alternative root. For the client J *is* the root, node K is node E, etc. and so it appears to the client as if the software had been installed with the actual root as J.

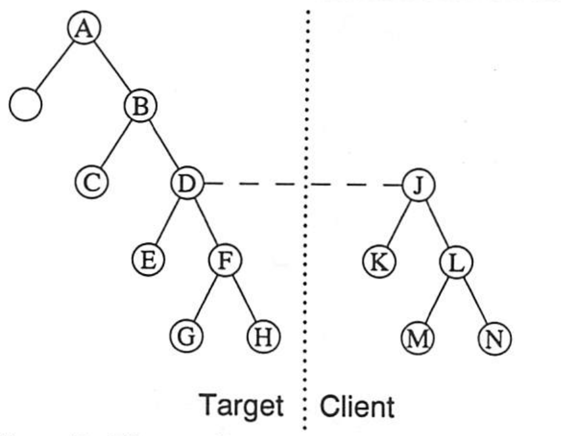

Figure 5: Filestore Structure for Discless Clients

Discless Clients

For discless clients the alternative root facility is obviously important, particularly for the operating system. Installation of other software only requires that the correct location in the (server) filestore is chosen for the software to be visible to the client. However, if the management information is to be visible to the client it is important that this too is located in the correct place. The provision of multiple catalogues is hence an important facility for discless clients.

Heterogeneous Management

The group would very much liked to have made the standard yield implementations that were interoperable at the task level. That is to say that the manager role could manage any of the other roles irrespective of the systems on which the roles were implemented provided they conformed to the standard. This would provide not only the capability of heterogeneous management using the commands defined in the standard but also a mechanism for enabling management applications to be written which could manage conformant systems. Unfortunately this would require the standard to refer to some mechanism for performing distributed tasks and no such mechanism is available as a *formal* standard. However, the group did receive several submissions specifying how this could be achieved using *de facto* standards. In addition some work was done on the formal definition of Managed Objects that corresponded to the definitions in the standard. An agreement has been reached with the X/Open Systems Management Working Group that they would progress this aspect of the standard, to be published in due course as an X/Open Specification.

How to Participate in the Ballot

When the ballot is about to take place (expected to be April/May 1994) the IEEE will advertise for participants. Anyone can submit comments but only those from members of the IEEE (or the Computer Society of the IEEE) are counted for the ballot; comments from others are "for information only". To ensure that you are notified of the ballot send your details to the author or the chair of the group, Jay Ashford at ashford@austin.ibm.com.

Acknowledgements

A lot of people have contributed to the draft standard, too many to be mentioned here. However, particular thanks are due to Jay Ashford, Matt Wicks and George Williams who have reviewed this paper to ensure that it reflects what the draft standard actually says rather than my own prejudices.

Author Information

Barrie Archer is a Systems Designer working in ICL Client-Server Systems. He works on the strategy of ICL's Systems Management products and participated in the development of ICL's OPEN*framework* Architecture for Systems Management. He is the ICL representative on the X/Open Systems Management Working Group and POSIX 1003.7 Working Group. He can be reached by mail on ICL Lovelace Road, BRACKNELL Berks, RG12 8SN, UK and electronically at barcher@oasis.icl.co.uk.

Managing the Mission Critical Environment

E. Scott Menter – Enterprise Systems Management Corporation

ABSTRACT

Mission critical environments present an entirely unique set of challenges to the systems manager. "Make it work and watch the budget" are the standing orders for most systems administrators, and yet in some situations the consequences of systems-related problems can be devastating, whereas in others the result is no more than inconvenience. As UNIX systems are increasingly at the heart of vital financial, medical, and defense technologies, we thought it would be useful to examine how systems administrators can cope with the challenges of managing such an environment. Finding and deploying the correct technology is only half the story: the systems administrator in this role is part of a social dynamic in which personnel and their skills are constantly being evaluated against their ability to support the mission. Thus, to be successful, the systems administrator has to be equally adept at handling both the politics and the technology that are endemic to the mission critical environment.

Defining the Mission Critical Environment

Every professional systems administrator works in a mission critical environment, to the extent that her job depends on the secure and reliable operation of systems and networks. Indeed, we expect that every systems manager reading this paper will recognize some elements of their daily lives, and therein (we hope!) lies the general appeal of our message. Still, something in particular is meant by the term "mission critical" as used by marketing organizations and want ads, so we'll try to come up with a clear definition and stick with it.

A mission critical environment is one in which even a brief interruption of service can have a serious effect on the ability of an organization to operate. We've used vague terms like "brief" and "serious" on purpose: there is a wide and fuzzy border between "critical" and "non-critical" environments. Still, like so much in life, you know a mission critical system or system component when you see it. We're reminded of the tiny transistor that recently failed in a NASA Mars probe, leading to the total loss of that billion dollar system. Another example might be a computer monitoring system located within a hospital's intensive care unit.

UNIX systems, once thought to be too quirky and unreliable for mission critical environments, can today be found in places that take security and reliability very seriously indeed. Among the earliest and perhaps most illustrative of these is the Wall Street trading floor, the site of billions of dollars of transactions each day. The ability of the Sun, HP, IBM, and other workstations to handle this load directly affects the function of the financial markets in which they operate, and therefore also the quality of life of pretty much everybody. Don't make the mistake of thinking that if the machines stopped working the old manual methods could be hurriedly put back into place: even if there were anybody around to remember those procedures, they could never cope with the volume of traffic associated with today's markets.

If you were to wander into a mission critical systems environment based on UNIX workstations and servers, what could you expect to find? We've noted some characteristics that are common to many of these sites:

- Beepers – In a truly mission critical situation, the staff (especially the systems administrators) may be called upon to perform some feat of technical heroism at any hour of the day or night.
- Attitude – Users have one. Managers have one. Developers have one. And systems administrators have one too. The business relies on the technology, and everybody's job is on the line. We've noted a severe sense-of-humor deficiency in many mission critical environments.
- Backups – Buying a piece of hardware? Better buy two, in case the first one fails. Oh, and what will we do if the network goes down? What about building power? What if sun spots trash our satellite connection to Tokyo? "No single point of failure" is the rallying cry of mission critical systems designers. Sometimes they add: "regardless of expense."
- Vendors – Lots of 'em. And they're hungry. They hear "regardless of expense" and come running. For systems administrators, this stampede has proved particularly frustrating as vendors in our market have been pushing

solutions that don't seem quite ready for prime time.

- Scale – Though not a necessary component of mission critical environments, many of these sites are deploying and managing systems on a large scale. And, most businesses will balk at simply hiring more and more systems administrators to handle the volume. The question in mission critical sites is how to manage large numbers of systems, often in geographically remote locations, without creating a huge demand for technical staff.

Into this situation wanders the happy (or perhaps hapless) systems administrator, who must learn to cope in a high-visibility, high-exposure situation while keeping his life – and his career – intact.

The Systems Administrator in the Mission Critical Environment

Our company is comprised of systems administrators (SAs) who spend their professional lives managing mission critical UNIX environments. We've noticed that just about all of our activities can be categorized under two headings: technology and politics.

It's easy to understand the technology part: after all, we're UNIX systems administrators, and we're constantly writing shell scripts, evaluating new hardware and software, inventing mechanisms for configuration management, etc. All of us got into this business because we were attracted to the technology, and we are making our living from our understanding of how that technology is best deployed and managed.

The need to be so involved with politics is less obvious to the non-SA. It can be readily explained, though. Systems administrators are in the business of managing expensive, important, and sometimes scarce resources. In many companies the allocation of those resources, like the allocation of profit and loss, is directly related to the political fortunes of the various parts of the business. How many times have we been approached with this sort of instruction: "You know those workstations that just arrived that George Jetson ordered for the engineering department? Well, we're going to 'loan' them to Barney Rubble at the sales department." The rise and fall of managers and organizations are often expressed in terms of who gets the new machines and who is assigned the more experienced technical support staff.

There is also a more subtle political aspect to the job of the systems administrator. Too often, the SA is faced with dueling priorities: conflicting tasks requested by end users. In a typical situation, one user will ask the SA to perform some function immediately, such as restoring a file from backups. Another user will ask the same SA to perform some other task immediately, such as clearing up a performance problem with a file server. In these situations, the systems administrator is suddenly called upon to make decisions that should be based on corporate priorities (about which he may be largely in the dark), but will most likely be based on who screamed the loudest, or which problem seems more interesting, or some other arbitrary basis.

Our intention in preparing this paper is to examine the special challenges of systems management in the mission critical environment. By viewing the situation from the dual perspectives of technology and politics, we hope to establish a reference point that will be useful to systems administrators currently working in this type of environment, and those who are considering moving to it. In order to do that, the rest of the paper will review some of the common functions associated with systems management – design, technology selection, deployment, operations, policies and procedures, and managing systems management – in the context of these two categories. The material represents an overview of observations drawn from our experience with various mission critical systems environments. Where possible, we'll present hints for preserving personal and professional happiness while working in the mission critical fast lane.

Design

Systems administrators are often involved in the design phase of infrastructure, applications, and certainly, systems management architectures. In terms of technology, there is real pressure in the mission critical environment to create redundant systems, systems that continue to work even in the event of a component failure. And yet, redundancy can often lead to overly complex designs that are actually *more* likely to fail due to their complexity, in spite of whatever failure modes were devised. Additionally, when legacy systems are included in the design, complexity is very difficult to avoid. In these situations, the use of a "gateway" system that isolates the points of contact between the new and old architectures can be very useful in keeping the design as simple as possible. In one of our current customer accounts we are helping to design redundant gateway systems between the workstations and the mainframes: the gateways themselves are usually vital components of a mission critical system.

Political pressures are certainly present in the design phase. Often, there is a fight to be involved in the design phase of a large project, or a desire to get "buy-in" from virtually every part of the enterprise. The systems administrators can find themselves left out of certain designs in which they ought to have early and extensive involvement; for example, new office or datacenter facilities, locally developed applications, or network architectures. Once the design is in progress, another important political question arises: is somebody being designed out of a job? Systems administrators have to ensure their presence in important design efforts, and show sensitivity to the question of whose livelihood may be affected by their plans.

Technology Selection

Typically, systems administrators will play a key role in the evaluation and acquisition of technology for the enterprise. In mission critical environments, the question of where to aim on the technology curve can be very important. On Wall Street, for example, individual departments were moving to abandon the security and comfort of the mainframe for the young and untested UNIX workstation environment in the mid to late 1980s, while MIS organizations in the same firms viewed this new technology as a threat. (They were right: many of those who fought hardest against the introduction of UNIX in the 80s find themselves in different jobs today.) Thus, the evaluation and acquisition stage can be a big mess of conflicting technical and political requirements.

From a purely technical standpoint, the important considerations today during this phase are *openness* and *interoperability*. Because definitions for these terms vary widely from one organization to the next, and because virtually all vendors will tell you that their products are bountifully endowed with both features, hitting the mark can be difficult for the technology selector. Systems administrators are usually in the best position to evaluate what will work best with what, as they tend to have a broader view of how software, hardware, and infrastructure interact than do other staff members.

The political dimension of technology selection can often be seen in the struggle over "build vs. buy" decisions. Particularly in mission critical environments, one often encounters a distrust of off-the-shelf software. If you're betting the business, as the reasoning goes, you want to be able to trust your applications, and (here's the flaw) the only applications you can really trust are those you build yourself. In truth, we feel that many companies have squandered millions on home grown solutions that could have as easily been implemented using software from the local Egghead outlet. In any event, systems administrators will be called on to make either solution fit into the environment, and will want a say in what vendors are reliable and what management features should be included in locally developed software.

Choosing the right vendor is as important as choosing the right product, of course. Carol Kubicki makes this point in her paper, [Kubicki92], "...[vendors] that might come into contact with your customers on your behalf should be held to the same standards as your own organization...[When the vendor fails to perform] the customer becomes more dissatisfied with the [systems] administration group who is ultimately responsible." The author is referring to field service personnel, but the same is true for any vendor, particularly in the mission critical environment.

Deployment

For some reason, the technology available for systems deployment seems to be behind that for other areas of systems administration. We assume that the reason for this is that each type of system has its own peculiar way of being installed: even various models of workstations from the same vendor can require different installation techniques. In the mission critical environment there is often a requirement to begin an installation after the users go home on Friday night and have everything reliably up and running before users return on Monday morning. As a result, the installation mechanisms provided by vendors are frequently insufficient to the task. In order to do the job well and in the brief time allotted, we have had to develop automated installation mechanisms that run *unattended*. The requirements for and design of such a system is worthy of its own paper: in this one we'll just point out that an unattended installation mechanism requires that all configuration information (names, IP addresses, etc.) be available in advance of the actual installation. The obvious reason is that if the machine is being installed without a person nearby, the answers to configuration questions have to be obtainable elsewhere. It turns out that recording this data in advance is a good idea anyway: installation time is the wrong time to be selecting names and configuring name service domains.

In the political arena, deployment is often the first contact between systems administrators and the ultimate end users of the machines and networks under their care. Because it's important that this first encounter be positive for all involved, it's vital that systems administrators be able to recognize a fully functional workstation when they see one. That statement is not as trite as it first appears. At installation time, there is often a great deal of successive approximation of workstation functionality. Here's what happens: the workstation is installed, the SA tests it, it seems to work (she can log in), and everything seems to be ready for the user. The user comes in in the morning and finds he can't log in. He calls the systems administrator, who discovers that the user isn't yet included in the NIS domain for that workstation, so she fixes that problem. A while later, the user calls back: he can log in, and most things seem to work, but he can't run Lotus. Within minutes, the SA has taken care of the situation, and is congratulating herself on responding so quickly to the user's problems. The phone rings: Lotus works, but when I try to print, nothing happens...

Operations

In contrast to the deployment phase, the operations phase has had considerable attention from vendors in terms of available software solutions. Lately a great deal of noise has been generated about the ability of systems management and network management tools to be *integrated.* This integration includes help desk tools, performance monitoring applications, problem tracking systems, and cable management tools, as well as the traditional event monitoring features normally associated with network management. While full integration of all these tools may be a desirable goal, the systems administrator has to be careful not to sacrifice functionality on the altar of systems integration. Especially in the mission critical environment, it's more essential that, for example, the network management tool be able to detect problems and generate alerts than that it automatically create trouble tickets in the problem tracking system. It's not that automated trouble ticket generation isn't a desirable thing; rather, the problem is that trying to get too many things working at once can delay the delivery of the whole system.

The operation of mission critical systems can be a high-intensity job. By definition (ours), these systems can't go down even briefly without adversely affecting the ability of the enterprise to stay in business. Political challenges aside (we'll get to that shortly), just making the technology work right can be a real effort. Therefore, the goals for managing configurations in this environment have to include *consistency*, *simplicity*, *predictability*, *auditability*, and *reversibility*.

- Consistency.
  In the mission critical environment, it's more important than ever that the same task is performed the same way every time. It's the rare site today that can claim that, for example, adding a user happens the same way no matter what systems administrator is on duty or what else is going on at the same moment. When you're busy it's surely tempting to toss a line into the `/etc/passwd` file, do a quick `mkdir; chown` and be done with it, figuring you can include the niceties later.
  Of course, "adding a user" may be too broad an operation to be done the same way every time. There may be derivative operations, like "adding a user to the systems administration group," or "adding a user to the sales domain" that qualify as tasks subject to the consistency requirement. The granularity can be adjusted to the environment, subject to the requirement of simplicity.
- Simplicity
  The importance of simplicity has been noted earlier in this paper; in this context, we mean that there should be a minimum number of configurations in the domain of managed objects. It's still hard to accomplish this long-understood goal. Workstations, for example, still have a small number of files that must be unique from machine to machine. Other types of configurations are frequently overlooked. At one site I've worked at there has been a great emphasis in minimizing router configurations, which generally have all sorts of uniqueness built into them.
- Predictability
  A non-deterministic system is clearly unsuited to a mission critical role. If, to revisit our earlier example, I invoke the procedure to add a new user, can I predict *exactly* what's going to happen? When we spoke of consistency, we talked about things happening the same way each time: predictability is about knowing *what* it is that's happening, and what the effects will be on the overall system.
- Auditability
  Well, we know what's happened, we know it happened the same way that it always does, but sometimes we also need to know *who* did it and *when* they did it. Also, we want to know what derivative operations or options were involved. In mission critical environments this ability is very important: identifying what happened and when can be crucial when the enterprise has been hit with a systems failure.
- Reversibility
  Sure, you know how to add a new user. What about removing that user? Do you know where all the files owned by that user are – even the ones that don't live in his home directory? What about mailing lists? Automount points? Any task that is performed has to be reversible, or the system will soon become unmanageable.

From a political standpoint, it is during the operations phase that things usually get hot. In a mission critical environment, everybody is quite serious about performance, reliability, and security. The systems administrator has to deal not only with the technical challenges involved in meeting those requirements, but also with the perception of the end users. During the famous Internet worm incident, the head guy at the site I was working at was extremely agitated about the possibility of "infection," in spite of the fact that we were not at the time network-connected. Knowing what your users are thinking is a very important part of getting operations right: that point is the major thrust of [Kubicki92].

It is also at this stage of things that users become very concerned with the organization of the systems management staff. We are big proponents of a modified centralized systems management model. Actually, our view is not so different from that of

Peg Schafer [Schafer93], despite that author's emphasis on the decentralized nature of her model. In fact, the model proposed in that paper is essentially a centralized one. The precise details of the structure we promote, and the differences between our view and that of [Schafer93] are left for another paper. The important thing to note in this context is that systems administrators in a mission critical environment have to be extremely sensitive to the immediacy of user requirements, and that those who are not may find themselves under relentless fire from the business. If a centralized group fails to be responsive to user needs, the result is not a nicely balanced structure like that presented in [Schafer93], but rather a complete dismemberment of the organization leading to total decentralization of the systems management function. This in turn can lead to increasing problems with security, standards, and efficiency for the business, and ultimately end in the failure of UNIX technology in an enterprise.

Policies and Procedures

Most systems administrators have the responsibility not only for maintaining hardware and software but also for helping to design the policies and procedures that form the day to day guidelines for running a large site. The important thing about policies and procedures in any site is that they be entirely consistent with that site's mission and objectives. In the mission critical environment, policies and procedures have to be developed that support and reinforce the vital nature of the operation: systems administrators have to be very flexible about how things work, because the mission may require quick adjustments to new situations.

From a technology standpoint, we've noticed something important: procedures that aren't encoded into software are not useful. Forget documenting your procedures: it's a waste of time. No two systems administrators will follow the same written procedures the same way every time. Procedures that are encoded into software can be powerful tools for keeping a large environment under control, while actually parceling out various bits of responsibility to non systems administrators. For example, if one could add a new user to their domain by simply typing

```
add_user lastname firstname machine
```

then it would no longer require a systems administrator to add a new user. Furthermore, the software could record who ran it, how often, and exactly what happened, for later perusal. And, systems administrators could confidently predict the outcome of each occurrence, because the function of the software is well known and trusted. In short, all the conditions we described earlier for reliable operation of a mission critical environment require procedures implemented into software in order to be fully met.

There is another important point to make about the politics of policies and procedures. As we've pointed out, they have to be flexible to accommodate changing business requirements. The important thing to remember, though, is that to the extent that ethical considerations are involved there should be no flexibility. There should still be a few things more important to the individual systems administrator than the mission of her organization. We mention this because it is all too common for businesses with mission critical systems to ask their systems administrators to take unethical actions in the name of defending the enterprise, and too often, the systems administrators involved don't even recognize that there is a moral issue involved. More education will help here, and we have high hopes that SAGE can further discussion on this problem.

Managing Systems Management

Much of today's technology is advertised as making it easier to manage departments and even entire enterprises. Middle management positions are falling by the wayside as technology enables widespread communication within and across departments, co-opting many middle management functions. Yet, one of the key supporters of this trend, the systems administration manager, is sometimes least adept at gleaning key information from his own systems.

An important feature of much of the world's technology is its ability to *informate*, a term coined in [Zuboff88]. The expression refers to the ability of systems to provide information as a side effect of their ability to *automate*. For example, when we develop and install custom software for our clients, we generally include a counter that tracks how many times the application is executed. As a result, we can demonstrate the value of the software to our clients by showing them a monotonically increasing usage curve. Similarly, many of the management tools used today by systems administrators can be exploited to provide valuable information. Want to justify the expense of that sendmail tutorial to the boss? Click, click, you've produced a report showing a steady increase in the number of email-related service requests submitted. Systems administration managers, like all other managers, have to use the best tools available to analyze their department's performance and to justify requests for resources, increased budgets, etc.

From a political standpoint, the recommendation of [Kubicki92] that customer surveys can be an important part of the management process makes a lot of sense. The issue of organization is felt most keenly by the systems administration manager, of course: demonstrating good service through system-generated information, survey results, and met commitments will go a long way towards supporting whatever structure the SA manager wants to put into place. Finally, and particularly problematic in mission critical environments, is the issue of junior staff. It is entirely common for junior staff to be thrown directly into a critical support area to sink or swim. In fact, just this approach was recently promoted by [Nather93]. It's the responsibility of the systems administration manager to make sure that junior staff receive the direction and support they require in order to succeed: nobody wins when one of these young SAs burns out or becomes disillusioned with his career. Messages of the "how can I get out of being a systems administrator" variety are all too common on Usenet.

Final Words of Wisdom

Although we hear a lot from academic sites about the work they're doing in this or that area of systems management, it turns out that some of the most useful and interesting stuff is being created and used in mission critical environments. The motivation to provide good solutions is simply higher at such sites. Of course, one also doesn't get exposed much to the results of these efforts, because mission critical environments are usually proprietary environments, and the business keeps its secrets to itself.

For systems administrators contemplating entering the world of mission critical systems, we have a bit of advice. Make sure a high-pressure, hard working environment appeals to you, and keep in mind that any chinks in your technical or political armor may be examined publicly and at length by those you are seeking to support. And finally, make absolutely sure *before* you accept any job that the enterprise at which you're interviewing has practices and policies that conform to your own ethical code.

Author Information

Scott Menter gave up the hectic but remunerative life of a Wall Street technical manager for the hectic but poorly compensated life of a southern California entrepreneur. His company, Enterprise Systems Management Corporation, provides technical and management expertise to companies with really large or widely distributed UNIX workstation installations. After getting his computer science degree from Brandeis University in 1985, he worked at various commercial and academic sites as a systems administrator, ending up in charge of the worldwide systems and network management department for Lehman Brothers. Finding few investment banks to work for in Orange County, Scott founded Enterprise Systems Management after he left New York in 1992. Today he splits his time between running the company and convincing Usenet readers that systems administration really is an okay way to make a living. Reach him electronically at escott@esm.com or telephonically at +1 714/573-4075.

Bibliography

[Kubicki92] Kubicki, C. *Customer Satisfaction Metrics and Measurement*, LISA Conference Proceedings, 1992.

[Schafer93] Schafer, P. "A Proposed New Model of Large Site System Administration", *;login: The USENIX Association Newsletter*, Vol 18, No. 1, Jan/Feb 1993, p. 18.

[Zuboff88] Zuboff, S. *In the Age of the Smart Machine*, Basic Books, Inc., 1988.

[Nather93] Nather, W. "Think or Thwim: The Cold Creek Approach to Systems Administration Training", *;login: The USENIX Association Newsletter*, Vol. 18, No. 4, Jul/Aug 1993, p. 22.

Open Systems Formal Evaluation Process

Brian William Keves – Systems And Network Management

ABSTRACT

System Administrators and Architects are faced with an abundance of products that can be used to solve a problem or fill a need. Some of these products are not truly compatible with existing or Open Systems hardware and software. They can even be proprietary solutions re-packaged to grab a slice of the "open systems" market. Separating the chaff from the true performers will continue to be an increasingly difficult problem.

This difficulty, combined with an increasing trend towards enterprise wide client/server technologies, shows a definite need to qualify procurement techniques and practices. This paper gives administrators the information to successfully organize or re-organize internal policies and procedures to perform appropriate evaluations of Open Systems products.

Introduction

Most procurement procedures and policies I have seen remind me of the following quote.

"At this point, I thought about the four horsemen of application/project development:

1. Conceived in Confusion
2. Born Into Ignorance
3. Developed in Chaos
4. Death by Neglect [1]"

Although this is not always true, many organizations need help with their entire procurement process. There are many aspects to this beyond simply picking up the phone and ordering equipment.

Small organizations usually don't worry about maintenance, compatibility, legal or purchasing issues. Large enterprises must pay attention to these details, since their volume is significantly higher from diverse internal organizations.

The problem is that many large organizations still handle the procurement process like a small enterprise. This results in the waste of major amounts of time and money. The main reasons for this attitude are:

- No Requirements
- No Communication
- No Standards
- No Coordination
- Internal Competition

At some point a large enterprise will look around and see a huge amount of equipment and software that will not work together. Most importantly, the information stored on these diverse platforms is not being shared, leading to unnecessary duplication and lost profit.

Every individual and team in an organization will benefit from a planned computing environment. To plan the environment there must be a coordinated procurement process. One of the major portions of procurement is the evaluation process. This paper will go into detail on the formal evaluation process, while still giving an overview of the procurement process and the organizational changes needed.

The main goal, as always, is to provide the customer (user) with the best available solution for the lowest cost.

Formal Versus Informal Evaluations

The difference between the formal and informal evaluation process is mainly in the documentation and scope of the task. Most people use the informal evaluation process, relying mainly on personal judgement and experience to architect a system or solution.

The formal process requires proper documentation and an unbiased approach to the evaluation of multiple vendors products. It also brings the customer into the focus of the procurement process.

Definitions

Following are definitions of terms used in this paper. Some of the terms are new and open to discussion, so it is best to be clear on the meanings.

Open Systems

This paper uses a market driven definition for Open Systems. If a solution can be obtained from five independent sources or is unique to one vendor but is demonstratively compatible, then it is considered "Open".

Downsizing

In conjunction with the trend to reduce the size of a company's workforce and occupied space, downsizing is used to describe the conversion from centralized to distributed computing resources. These tasks are occurring hand in hand and this paper uses downsizing to indicate the trend towards distributed client/server environments.

Customer

This is a generic term applicable to all enterprises, including commercial, educational and research organizations. It mainly refers to the user of computing resources that a support organization must architect and administer.

Profit

Like the definition for customer, profit is being used to indicate success in an endeavour, however success is indicated in the enterprise.

Standards Requirements

It is very important to understand what standards need to be employed within the enterprise. Europe is dedicated to using formal standards for all products and services. ISO is the main example of this. In the United States there are many formal and informal standards. An example of an informal standard is Sun's NFS. It is an industry or de facto standard that was not developed by an official standards body. [2]

The choice of standards is important to the successful conclusion of the procurement process. Research needs to be done concerning what standards an enterprise must adhere to.

Methodology And Test Standards

There are a number of methodologies available which will help with the procurement and formal evaluation processes. Some standards bodies are also writing formal specifications.

- COS User Requirements Process [3]
- FURPS [4]
- IEEE P1003.0 [5]
- IEEE 1003.3-1990 [6]
- ISO Technical Report 10000-1 [7]
- SILC [8]
- X/Open Open Systems Directive [9]

Why Perform Formal Evaluations?

As Administrators are increasingly thrust into the position of architecting and purchasing recommendation and authority, they realize that they are required to provide more justification for a large purchase than "it is the right product to buy." Management wants assurances that the products they are being asked to purchase are going to work now and in the future.

"Re-tooling" on the workstation level is much more expensive in a large enterprise than the old centralized approach. For example, a change to an operating system can take days to propagate throughout the network.

Financial

A competitive edge translates into profit. Companies have an obvious bottom line motive, but even educational and research institutes employ the concept of profit. Profit comes from the ability to succeed better than the competition, whoever it is.

Information is becoming one of the most important aspects of life today. Organizations that can manipulate and use its information faster and more accurately than others will be the most profitable. Organizations that have forgotten the profit motive and let politics dictate their technical decisions are finding themselves left behind. Downsizing is more than just a trend. It is a direct result of large companies ignoring their infrastructure and finding they can no longer do business.

Legal

This is a subject that many individuals in our field never think will affect them, but they may be in for a surprise. Many large companies, universities and research institutes work on government projects. The requirements for these projects are usually more stringent than in industry, as are the consequences of failure. Sometimes joint partnerships or agreements can also lead to legal complications.

When projects fail, no matter who was to blame, reputations can be damaged, companies can go out of business and grants can be lost. It is not necessary to dwell on this aspect, but a formal evaluation process will allow you to show a paper trail of your activities. This documentation is useful for other things, like keeping management informed and interested.

It is not unknown for legal disputes to take years to resolve. It is better to have written proof of activities than to rely on memory.

Professional

Our profession is no longer a part time afterthought. We are directly responsible for the productivity of a large amount of users. Their main goal is to perform their work. We have a responsibility to those users to maintain the highest levels of competence and professionalism. Enterprises are growing exponentially and we have to keep up.

What To Do First

Organizations need to become aware of the need and the money savings of proper procedures. This can be achieved through a two pronged approach. First, put the policies and procedures in place for a subset of the total enterprise. The final step is to publicise the success of the new approach, so other organizations will wish to join in.

Policy

Obviously, management buy-in is critical to set and enforce policy changes, but the user community, the real customer, must not be ignored during this phase. They should help with setting policy, just as they should help with establishing standards and specifying requirements.

The earlier that policy is introduced into the life cycle of an enterprise, the faster the results will be apparent. Ultimately, a new venture should have all procurement and administrative policies in place before a single piece of equipment is evaluated and purchased.

Procedures

Flexibility is the key! The goal of procedures and policy is not to create unnecessary red tape, but simply to make an organization smarter in the way they deal with the computing environment.

While enterprise wide policy is being established, the procedures necessary to operate can be put in place for all aspects of the procurement process, at least on an interim basis. Many times these "interim procedures" can very quickly become permanent, so it is wise to make the procedures easy to follow and complete from the start.

The first step is to get the customer involved with the procurement cycle by encouraging them to plan for future requirements. Since a formal evaluation cycle can take 3 to 6 months, it is important to mold customer attitudes and expectations. It can actually by detrimental to present the support organization as a source of immediate wizardry. This falls apart when enterprises grow beyond a certain point.

Next, research a formal evaluation process that is comfortable and design the applicable procedures for the enterprise. All formal standards and known requirements should be included and frequently reviewed for updating procedures.

Completion of the process include establishing liaisons and procedures for dealing with Legal and Purchasing. The goal here is to speed the evaluation contract and purchasing processing.

Formal Evaluation Process

This is a description of a strict, formal evaluation process. This should not be conceived as an immediately implementable goal, but as the desired end result of the process of change within the enterprise.

Requirements

As mentioned above, users and management need to be involved in future requirements planning. A yearly planning cycle seems to work best in today's high-tech enterprises. This does not mean that support organizations should ignore requirements and needs that occur out of cycle. Procedures and budgets should be established with the expectation that emergencies and changes to requirements will occur. If the policy and procedures work, then support organizations will have the time to implement the unexpected. This is preferable to the resource intensive, short cycle, non-communicative planning that seems to be the norm.

Request For Proposals

Whichever the preferred name, Request For Proposals (RFP) or Request For Quotes (RFQ), this document is the corner stone of the evaluation process. While the procurement process can skip a formal evaluation if needed, a formal evaluation cannot exclude a request to multiple vendors.

The RFP is absolutely the most important document of an evaluation. This is where requirements and the local environment, as well as existing standards, are communicated to a number of vendors who can satisfy the request. It is a good idea to communicate the RFP widely, as this will provide a large base from which to choose a small number of evaluation units.

A complete RFP should consist of the following sections:

- Purpose
- Terminology
- Business Requirements
- Functional Requirements [2]

An important point to mention is to make sure that the RFP reflects the organization's and user's requirements exactly. Do not "pad" the RFP with irrelevant standards and requirements, since this can make it difficult to succeed in finding the right product for the task.

Research

It is important to research the various aspects of an evaluation. Venders need to be found, products reviews obtained and colleagues queried. Many sources of information are available, in fact too many to read constantly:

- Trade Magazines
- Product Directories & Guides
- Internet - News, Mailing Lists and Databases

User Group

At the same time the RFP is being written and research is taking place, a small group of users should be assembled on a regular basis to do the following:

- Help Define Requirements
- Suggest RFP Recipients
- Review RFP Responses
- Form Evaluation Teams
- Recommend Final Product

When properly organized the users will do most of the work for the entire evaluation. It is the responsibility of the support organization to facilitate meetings and properly document the results as well as set and apply policy and procedures.

Properly documented, the results of the user group's participation will provide most of the information necessary to ensure management buy-in. It also encourages users to attempt to solve problems in the organization instead of constantly pointing the finger of responsibility.

Paper Evaluation

Once a suitable number of responses to the RFP have arrived, a paper evaluation is needed. This is the preliminary cut based on the stated capabilities of the product. This is where the research previously mentioned should be used to ensure that the stated capabilities are accurate. This will save the time of doing a physical evaluation on a product that has a known problem or does not really satisfy the requirements.

Physical Evaluation/Testing

Once the list of potential vendors has been reduced to a manageable number, arrangements need to be made to obtain an evaluation product from each of the selected vendors. This should be the current version of the product and not a demo or presentation version. The RFP should stipulate that a 60 day evaluation of the product will be necessary, thereby notifying the vendor of the need to be prepared for this eventuality.

Testing for conformance to the standards and requirements is upmost. Second, testing for compatibility with existing equipment.

The user evaluation teams should be given clear direction concerning the length of time they have for the evaluation and if the product is large enough, split up the evaluation tasks to different individuals or teams. Each individual must give feedback on the evaluations.

Use the vendor's technical support to solve problems and answer questions that arise during the evaluation. This is actually an evaluation task that can easily be accomplished. Workarounds and fixes may be needed, but this should not necessarily invalidate the evaluation. Care must be taken to properly configure and maintain the evaluation product.

One hint that will save lots of trouble is to ensure that the users do not perform unreproducible production work on an evaluation product. This means restricting the access to the product to the evaluation team and establishing a clear policy against this behavior. Sometimes, customers are eager to use a new product and will compromise the evaluation by insisting that only the one product they are using can be purchased, which invalidates the reason behind the evaluation, which is to find the product that is best for all users.

Feedback And Consensus

Giving feedback to the vendor during the evaluation is an acceptable practice. It helps the vendor to fix problems and re-direct misconceptions about the product. But don't give them information concerning their competition's product. This can be viewed as unprofessional behavior.

Make sure that the evaluation teams give the appropriate feedback needed to make a decision on the product. An evaluation form is the easiest way to accomplish this, possible with a reward as an inducement to complete it. The reward will depend on the situation, but this is a well accepted technique that ranges from candy to bonuses.

Cleaning Up And Documentation

The last step in the evaluation process is to make sure that all unpurchased products are returned to the vendor within the time frame of the evaluation. Some vendors will bill for the product once the evaluation period has expired.

Collect all documentation on the evaluation into a report. Present this to management and the user group concerned with the evaluation with a summary explaining the reasoning behind the current purchase.

Publish a regular newsletter, in printed or electronic form, which keeps the general user up to date on new products and successful evaluations.

Benefits

Most support organizations are cost centers, not profit centers. Therefore it is important to save money whenever possible. This will allow better utilization of equipment and administration budgets. This can translate directly into bonuses and favorable performance appraisals.

Return On Investment (ROI) is also an important concept in business. Making the most of the money spent for Open Systems technology will help determine the long term success of a company. A formal procedure will help to ensure a proper match between requirements, cost and user performance.

Summary

The need for definitive Open Systems evaluation methods is apparent from the current state of the industry. This paper is an attempt to convince the reader to start applying formal techniques now, before problems occur.

Organizations will live and die on their ability to manage the phenomenal growth in information and technological solutions.

Acknowledgements

I would like to thank Ross Baker for pointing me to industry methodologies and reviewing my outline. I would also like to thank the Usenix reviewers for their input. Last but not least I thank my fellow consultant John Benton for his inspirational and quotable homilies.

Author Information

Brian Keves runs his own national consulting firm and has had over eight years of experience in architecting, implementing and managing systems and networks based on Open Systems Client/Server standards. Some of Brian's past clients include Boeing, General Atomics, Mead and The University of California San Diego. He is currently architecting and implementing a nationwide Community Health Information Network for Ameritech. Brian can be contacted at Systems And Network Management, P.O. Box 1819, Oceanside, CA 92051 or via e-mail at keves@Sanm.COM.

References

[1] J.T Benton. "30 Years With Computers (And Other Narrow Opinions)." Executive Information Development Company, Barrington, Illinois, 1991.

[2] K.M Lewis. "Standards-Based Procurement Using POSIX and XPG." Uniforum, 1993.

[3] User Alliance for Open Systems (COS). "User Requirements Process." Ed Albriggo, COS.

[4] D.L. Casewell, R.B. Grady. "Functionality, Usability, Reliability, Performance and Support Ability Software Metrics: Establishing A Company Wide Program."

[5] IEEE P1003.0 "Guide to the POSIX Open Systems Environment." Kevin Lewis, Digital Equipment Corporation.

[6] IEEE 1003.3-1990 "Standard POSIX Test Methods." Lisa Granoien, IEEE Computer Society.

[7] ISO Technical Report 10000-1 "Framework and Taxonomy of International Standardized Profiles." Clyde Robichaux, AT&T Corp.

[8] "Systems Integration Life Cycle Methodology." SLH Systemhouse. Unpublished.

[9] X/Open "Open Systems Directive."

A Case Study on Moves and Mergers

John Schimmel – Silicon Graphics Inc.

ABSTRACT

This is a case study on the issues surrounding a large administration group move or merger. The actual case is that of the Silicon Graphics Inc. merger with Mips Computer Systems Inc., but the lessons learned are global to such events. The paper goes through the merger process, and the subsequent move of the new Mips engineering organization, pointing out many of the key decisions and problems that were overcome.

Introduction

In July of 1992 Mips Computer Systems, a small, high-tech, computer company was purchased by one of their best customers, Silicon Graphics Inc. At the time Mips was comprised of approximately 700 people designing RISC computer processors, and systems. The company used over 1400 computer systems spread over approximately 50 networks. By the end of September the six building in the Mips facility were empty. All of the remaining employees had been merged into the main Silicon Graphics campus along with their computer systems and labs. The administrators that carried out the move had to reinvent many solutions that other groups had learned before, and all systems administrators can learn from their mistakes. This paper documents many of the problems and solutions that came about from this project in the hopes that it will aid the next such attempt.

The Project

The upper management of Silicon Graphics quickly made the decision that centralizing all of their projects onto the main company campus was a key to the success of the merger. Mips was in the middle of several important projects, and SGI did not want to lose key Mips employees due to the feeling of neglect or mistreatment. At the same time, SGI did not want to seriously effect the engineering schedules on projects continuing within Silicon Graphics itself.

In addition to the merger, Silicon Graphics was in the process of a major restructuring which included a massive reorganization of their campus. This greatly complicated the progress on finding locations for the Mips employees moving into the Silicon Graphics facility.

One large part of the project was the creation of a new company which contained all of the processor and compiler development for the Mips architecture. This included opening a new building on the SGI campus containing a new network design, offices, labs, and a computer room.

Beyond the physical plant for this new company, Mips Technologies, Inc., was the creation of a new sub-domain under Silicon Graphics, mti.sgi.com, with its own mail, news, and name services. This new sub-domain needed to work within the SGI domain in order to facilitate joint engineering projects, but also had to remain entirely separable logically.

A major part of the project was the incorporation of nearly 700 new Unix accounts into the SGI name-space. Over 200 of the users from mips had to change their login names, and nearly all of them had to change their user IDs to fit into the flat name-space on the SGI campus. This required the changing of ownerships on millions of files on the Mips computer systems that were moving to the new site.

Dozens of other projects were simultaneously underway to merge the database systems from the new entities, to deal with separate bug and information tracking systems, to separate the compute resources for the Mips engineering projects, to centralize source and release trees, and much more.

Early Concerns

The immediate concerns after the announcement of the merger were mostly concerned with the structure of Silicon Graphics, and how they were going to fit the Mips contingency into it. At the same time as the merger with Mips Computer Systems, Silicon Graphics decided to have a major reorganization of the current divisions, which delayed the question of Mips until after that had succeeded.

Once the decision to create a separate company containing all of the VLSI and compiler design had been made, many personnel decisions needed to be worked out about which support people would move over as part of the MTI organization, and which would be merged into other portions of the Silicon Graphics engineering department. The members of the Mips support group were all expected to apply and interview for the positions in Silicon Graphics at the same time that they were trying to support projects at Mips, design the domain for MTI, and learn the new operating system and hardware platforms in order to train the Mips engineers.

In designing the new domain the systems administrators from Mips needed to know how the hardware from Mips was going to fit into the compute philosophy at SGI. Were they to move over all of the workstations, or was Silicon Graphics going to attempt to provide engineers in certain departments with an SGI platform to work with. Were they going to bring over the hardware labs from Mips or were those projects going to be canceled. If they were to be in a new building how much of the network hardware was going to be brought over from the Mips facility, and how much was going to be setup ahead of time. To each question there was an answer, but rarely did the answer come before the implementation was begun.

There were a lot of concerns about merging the projects themselves in the beginning as well. Both Mips and Silicon Graphics had a compiler project, and they each had VLSI groups. They had two different operating systems groups, and various systems projects. Some managers wanted an immediate merger of the two projects while some wanted this delayed as long as possible so that current projects would be less effected. For instance, the compiler organization wanted a single group on day one, while the VLSI groups wanted to delay this until the follow-up projects were organized. Each of the merged projects had computer needs that had to be considered separately.

Pre-merger Decisions

There were a lot of decisions made on how to implement the moment of the move that had lasting consequences on how things were approached down the road. This is a discussion on some of the more important of these.

Probably the most important decision that was made was that the actual merge of the environments would not happen until the day an engineer moved from one site to the other. It was decided that this was the right thing to do to least effect the projects at either site. The number of employees sitting in Mips buildings and working on projects at SGI was relatively small, as was the reverse, and these employees could maintain separate accounts in the opposite domain. The consequences of this decision were vast. This meant that the User ID for accounts would not change until the day the user moved. Real work over NFS was disallowed across the boundary because this would inevitably cause confusion with UID/GID space. It also meant that the domain name would suddenly switch on move day from mips.com to mti.sgi.com or some other domain, and all of a users mail had to be forwarded. It also meant that there would be a small number of machines that would have to appear to be in both domains for a short time in order not to break dependencies within certain projects.

Another relatively important decision was that the network layout would be totally redesigned for the new site. This meant that every machine that moved would have to change addresses on the move, that licenses would all have to change when the machine moved, and that network boards would have to be shuffled between machines during the physical move to get reasonable network gateways configured.

A third long-lasting decision was that the mti.sgi.com domain was going to be treated differently from the other domains on campus in order to delay as much of the pain of relearning the environment as long as possible. This included such things as trying to support BSD and System V printer spooling to all the printers in the domain, to support NIS and a centralized rdist distribution for machines, to support the Mips mail configuration, and the mips.com mail aliases indefinitely, and to support the Mips bugs database. In other domains the old mips workstations would be reconfigured to run NIS, their mail configuration would change dramatically, their users would have to learn how to use the System V printer mechanism immediately, and many other things. It was the decision of the Mips engineering managers that they would like to try to delay these changes until after their current projects were completed.

Physical Plant

The new company, Mips Technologies Inc., that was started out of the merged organization was moved into a new building on the Silicon Graphics campus. Planning the physical plant represented a large part of the work in creating the new domain. This is a discussion on some of the decisions made during the creation of MTI, and their results.

One of the early battles was over the existence of a raised floor computer room. There were a lot of questions from upper management about the need for special conditions in which to place the MTI computer systems. There was only one other computer room in all of SGI before the Mips arrival, and this was for the MIS mainframe systems. All the rest of engineering just had machines sitting in labs or in peoples offices, and managers were under the impression that MTI could follow the SGI engineering example to save space and cost. The problem was that MTI had several hundred server machines of various sizes, and all of the systems administrators on the project knew that there was no alternative to a real computer room to house these machines. It took a great deal of effort to have a computer room installed in the MTI building, and to prove that the Mips servers could not be dealt with in any other way.

Another piece of the puzzle was the computer network physical wiring. MTI used ethernet over twisted pair to two central closets. This included more that 600 lines of twisted pair home-run into the computer room. All of the cabling was standard level 5 twisted pair. The cost of the wiring plant was brought up several times, but most of the managers had seen wiring plants done inadequately and did not put up too much of a fight. The only weakness in the wiring plant as implemented was that the MTI building was not tied heavily enough into the surrounding buildings. The MTI organization quickly spread into the building next door, but there was not enough fiber between the buildings to have more than a couple of nets spanning both buildings, and being tied in to the MTI computer room. Very few other networking options were considered for this project, use of higher speed networks was delayed for a reasonable alternative, but the wiring plant was designed to support the more promising networking possibilities.

Other physical plant questions that were raised included the amount of lab space that was required for each of the engineering and support groups, the locations for printers, copiers, and other centralized resources, and, of course, layout of office locations. These were primarily political questions that were just decided by committee in a heavily attended meeting, but they required systems administration presence, and cut into the time for other preparations.

Move Day

The actual physical move was relatively easy, but there were an enormous number of problems that were found at that point for which some pretty creative answers were found.

The move was done in three phases. In some ways this was nice, but for the most part it made it even more difficult to not interrupt the current projects. Phase one was moving the Silicon Graphics processor group into the new building. This was the simplest of the move phases since this group was already on the Silicon Graphics campus, and was already used to doing things the SGI way. The tough parts of this move were trying to get the domain mti.sgi.com setup in advance, testing all of the network connections, and bringing up the machines in their new network configuration. Little else had to change for these people to do work. However, there were many problems encountered that had to be addressed.

The first was that there were several license servers that did not understand domain names, and do not work if they do not have aliases in the local NIS database.

Another was that the Sun OS machines expected to receive a non-qualified address from gethostbyname() when they booted to set the right hostname for sendmail, while all of the other machines wanted a fully qualified name.

And, a third was that there were some extreme bugs in the routers that were purchased for the new network implementation such that if you plugged in a machine setup for the wrong subnet into some network the routers stopped forwarding packets.

In addition to the technical difficulties there were the weaknesses in experience with the Silicon Graphics systems to overcome. Many of the problems in the initial move were that the people on the move team just didn't know were to find things in the Irix operating system, or how to jumper boards that had to be reconfigured. Many of the default configuration decisions were different in Irix so that much of the automation did not work as planned.

Phase two was moving the compiler group into the new building. The compiler group was made up of people from Mips and from SGI, and created a considerable bit of difficulty. The biggest issue was that the Mips compiler group and the Mips operating systems group were sharing source trees, and working together very closely, and wanted to be able to work as usual without too much effort until the tapeout for RISC/os. This meant that the User IDs for these compiler people were not changed at this point, and they continued to use the mips.com domain for their machines. The named configuration was heavily messaged so that it would look as if every machine at mips was in the mti.sgi.com domain when named was queried from the Silicon Graphics side of the pipe, and that every machine in MTI was part of the mips.com domain when viewed from the mips side. This made it possible to hide the fact that source machines were moving away from Mips without changing hundreds of links, and breaking dependencies.

Another interesting part of this move was that all of the customization was automated by a script that was executed on bootup at the new site. This script setup the new host address, and copied over a large list of configuration files. The lesson was quickly learned that what these types of scripts should do is to copy over a new script, and then just execute it. The customization script changed about ten times during the afternoon as machines were brought up and problems were found.

The third phase of the move was the largest, and most complicated. In this phase they moved all of the Mips VLSI groups, and revisited all of the machines from the Mips compiler group to change the domain to mti.sgi.com. Nearly five hundred machines were moved in this one phase which made it a logistic nightmare. But, thanks to the earlier efforts in writing automation scripts to do most of the work, this was not too overwhelming. For most machines the customization was done almost entirely by the script, and all that people on the move team had to do was run around and turn machines on, and make sure that they came up and ran the script without errors. There were still some problems in this move that made things difficult.

The biggest problem actually came about because of some bugs in the program that was written to remap the UIDs on some architectures of machines, and more importantly a few typos in the fix that resulted in large numbers of OS files getting there permissions and ownerships changed.

Another problem that came up at this point was that they had chosen this time to switch the mips.com MX record over to a machine at SGI, and that the new mail gateway had not been correctly configured. Two days of bounced mips.com mail resulted, and annoyed many of the Mips engineers when they came back to work.

Another issue that came up at about this time was that many tools internal to SGI did not deal correctly with subnetting, and made it very difficult for the programmers who had moved into MTI to do any reasonable work. Of course, much of this had to do with the fact that the software people in MTI were upgrading to new alpha versions of the Irix operating system at the same time as systems administrators were trying to deal with move problems.

A final interesting issue comes about primarily due to network design by committee. The systems administrators in MTI did not fully understand the network traffic flow for VLSI work so they called together meetings with CAD engineers in Mips to try to layout a reasonable network for the new site. In the meeting it was decided that the structure of how they were designing, and their data location model, was going to have to change anyway, so they might as well plan for the new model instead of changing things six months after getting into the building. The problem with such choices is that changing the network layout is easier than getting people to change how they work, and the transfer to a new compute model took months, with all heat from network problems during that transition being transferred onto the systems administration group.

Politics

The biggest problem in dealing with a large merger proved to be dealing with the merger of culture and organization structure. A couple of good examples for this came out during this merger, and in the months following it.

The most difficult issue for the systems administrators to handle was the changes in the systems administration model itself. Mips had a centralized systems administration group that handled all aspects in a single group. Silicon Graphics was driven by project, and whatever projects felt they needed help hired a person to do support for their group, and the rest just had engineers supporting their resources. Mips had central servers and dataless workstations that had all of their files updated centrally out of rdist. Silicon Graphics followed a peer model where people had their homes, and all of their important data on their workstations, and all of the machines on the network were highly customized by the people who used them. Mips followed relatively strict rules for internal security of the machines while SGI had the iron door philosophy where the gateways are very secure, but all the rest of the machines are basically open. Mips systems administrators maintained the engineering hardware, while the customer service department maintained the hardware platforms on the SGI campus. Plus, of course, about two hundred other major differences in administration philosophy.

The structure changes were also hard to get used to. Mips had a very simple management hierarchy. People worked together well, but each person still had one boss, and had to please a single individual. SGI has more of a matrix management, where each person has a boss, but also reports to any other manager that he directly influences. This might be reasonable for an engineer, but for a support person this means that an individual may have a dozen indirect managers.

Mips had a single domain, and a flat namespace throughout the company maintained by a single group within the company. Silicon Graphics, however, is highly subdomained, and the interaction between these domains can be very difficult to understand. For instance, the MTI systems administrators can plan their networks, but they have almost no say in the implementation of the plan. The network hardware and wiring is all handled out of another group in a different domain. Also an engineer can have accounts in multiple domains, and they are each treated as separate account with little automatic updating in configuration. If a user wishes to change his password he must change it in each of the domains were he has accounts separately.

One other issue that made this merger more difficult than it should have been was the whole issue of having employees, that should have been concerned with administrating the move, spend large portions of their time trying to locate a permanent position within the new organization. Many of the people on the move team were temporary employees that had no assurances of having a job after the move was completed, and nearly all of the people had to work on finding a comfortable new position in Silicon Graphics, and proving their capabilities to a new set of managers.

Conclusions

The Silicon Graphics Inc., Mips Computer Systems Inc. merger and subsequent move was a beast unto itself, but there are many lessons that can be drawn out of the trials involved that will help in other such instances. This is a summary of some of the clear examples.

The first hint is to automate as much as possible, but assume that the script and the configuration files will change during the move. During the MTI move almost none of the dozen or so configuration files that were copied around went through the entire process without change.

The second is to never assume that the people who setup the machines will know anything about systems administration. Do to the enormous scale of most moves, the systems administrators typically end up doing only a small part of the work on the day of the move. Everything should be setup so that the machines just need to be plugged in and turned on for any changes to take place.

A third would be to make sure that all machines trust at least one host from a central location during the move. All of the machines within MTI were setup to trust the domain master mti.mti.sgi.com during the move so that changes in the configuration could be easily pushed around from a central host. This is especially important if the move is multi-phased since configuration files are almost sure to change as each step is achieved.

Another hint would be to carefully layout the network maps well in advance of the move. Every team member should have a clear map, containing every host, so that they can easily see how things fit together. Many of the initial problems on move day will be connectivity related. Such problems as a misconfigured gated on a server that ceases sending out route changes an hour into setup are common.

A fifth point is that order is important on bringing the machines down, and in bringing them back up. This is a relatively obvious point to a systems administrator, but not to a mover or a facilities person. If the script that runs when a machine comes up cannot get to a host that it wants to copy files from then bad things happen.

Another obvious point, but one that is often neglected in the panic of a move, is that everything should be documented, and hard copies should be available. The host file, a network map, a copy of each of the configuration files, directions in setting things up, etc. should be printed out and given to each person on the move team. The basic point is "kill a tree; save a systems administrator". And, part of the same point, all cables, outlets, etc., should be carefully labeled as things are setup.

As difficult as moves and mergers are for systems administrators it is important to remember that every person in the company, or building, is being affected, and calm heads prevail. Stress does interesting things to people, and it should be considered in the overall plan. Planning to have meals taken care of in advance, or staggering schedules, can help more than all the technical planning in the world. And, in the same vane, everyone should be prepared for the possibility of catastrophic failure. If the supported engineers have unrealistic expectations of being able to come into work the day after the move, and having everything work perfectly then conflict is inevitable.

Another idea is to warn all of maintenance contractors that you are planning to move hardware so that they can prepare a stockpile of parts that commonly fail in a move. Some maintenance agreements have special clauses with respect to moving hardware that necessitate the contractors presence, and possible additional payment.

Setup a report mechanism in advance. Leave a form that people should fill out for move problems sitting in each chair, and then deal with the problems as people turn them in, checking each one off as it is done. Giving a clear impression that problems are being addressed is very important.

An important hint is that there is a priority to the services that must be up quickly, and without fail. The hottest topics in MTI at the time of the move were news and mail. Users were relatively happy if they could come into work, sit down at their workstation, and read news, even if they were unable to work on their projects.

Don't neglect the little things. A common reaction to the stress in a large move is to ignore the small problems to concentrate on the major issues. To an engineer there are no minor problems, and the fact that his workstation is facing the wrong direction is just as important as the fact that another system can't talk on the net. It is well worth having a person dedicated to these types of problems.

The other people in the building want to help, and often feel powerless sitting around in their office all day watching systems administrators jogging back and forth down the hallway taking care of problems. It is a common impression that engineers, or managers, would only confuse the issue and make things more difficult in the long run, but there are things that an engineering manager can do to be helpful. Having them put on a barbecue on move day, or running to the store to get powerstrips are good examples.

A good thing to remember is that the administrators should be single-minded about making the move successful. None of these people should be allowed to worry about anything outside of the task at hand. The added pressure of having to locate their next job, or their next meal should be removed.

There should be a central point of communication and status for the move team. At Mips this was a giant whiteboard with a constantly maintained list of move issues that needed to be dealt with, and what their current status was.

Any third party orders need to be sent out should be taken care of very early. This includes primarily hardware orders for wiring, network equipment, new machines/printers, etc. But, it also includes things like licenses that may have to migrated. Most third party vendors have trouble accurately estimating delivery times.

Always remember, this is only a job. Any major change in a company is a carefully measured risk, and falls back on higher management. Moves and mergers are one of the most difficult and stressful of all systems administration projects, but they can also be one of the most fun.

Author Information

John Schimmel was educated at the University of California, San Diego, where he was introduced to the world of systems administration. He has been doing systems administration work for six years, and is currently a Senior Programmer/Analyst for the Engineering Computer Services group of Mips Technologies Inc. Reach him via U.S. Mail at Silicon Graphics Inc. M.S. 10U-178; 2011 N. Shoreline Blvd.; Mountain View, CA 94039. Reach him electronically at jes@sgi.com.

Guerrilla System Administration: Scaling Small Group Systems Administration To A Larger Installed Base

Tim Hunter & Scott Watanabe – University of Colorado at Boulder

ABSTRACT

Many university sites have a large number of machines that consist of a variety of platforms and operating systems. Because of limited budgets, many use a small number of full-time systems administrators supervising a group of undergraduates to administer the campus network. The usual solution is to have the full-time administrators act as managers for a pool of junior and senior undergraduates. The junior administrators learn on their own, with limited help from the senior administrators and managers. This model works well for a small group of undergraduates who can remain in close contact. However, recent increase in demand for system administration requires hiring more undergraduates, causing the small group system to break down – junior administrators do not get as much contact with senior administrators and management, and some lose touch altogether.

This paper presents modifications to the small group model intended to improve training of junior administrators, and associated changes to improve customer service. The driving idea is to give undergraduates as much authority as they can handle responsibly. This means making current undergraduates a part of the interviewing process for new administrators; taking advantage of the benefits of the small group model by forming small groups of junior administrators lead by senior undergraduates, who receive minimal guidance from management; and creating standing orders that extend the authority of a staff member in a crisis situation. These improvements, as a whole, have worked very well. New hires take less time to become part of the team; small groups of undergraduates provide better support for junior administrators, and free full-time administrators from dealing with the details of a large project; competent junior administrators are able to handle most situations without help.

Introduction

UnixOps is a division of the University of Colorado at Boulder's Computing and Networking Services department. Our task is to provide network and system administration services for UNIX systems to paying customers, most of whom are in the College of Engineering. We also receive general fund money to perform a number of campus-wide services including BIND, mail, source access, door card access, and USENET news. We also maintain host, user ID, and mail alias databases for the campus. Our operation is more like a "real-world" business than an exclusive service-provider whose income only comes from one source. However, we do operate in both realms.

UnixOps has been in existence in one form or another since 1981. It was created to maintain a few workstations in the Engineering Center, and has grown exponentially since then. About five years ago, there were approximately as many staff members as there were machines to administer. Now we provide full support for six large computing labs and twenty-five smaller labs and stand-alone machines – roughly two hundred machines in all. We provide network support to fifteen other hosts. While most of the machines are Sun workstations, we also administer DECs running Ultrix, Silicon Graphics, and one HP Snake. Most of these machines are in the engineering center, but there are also hosts in several other buildings on campus: math, geology, economics, and physics.

Currently our staff consists of three full-time professionals, who split administrative and management duties between them. We have a systems staff of fifteen students, along with three students to do tape backups and one to assist with administrative work. We further classify the systems students into "seniors" and "juniors" (not to be confused with that student's status in school). The former have usually worked at UnixOps for more than a year, and can handle nearly all tasks without supervision or assistance. The junior administrators are still on the learning curve – they may have worked nearly as long as the senior administrators, but still require some assistance (or at least assurance that their solutions are valid) on most assignments.

Small Group Systems Administration

Though the staff has recently expanded, the proportions – one professional for every five students

– have remained the same. Until a year ago, our staff consisted of two full-time administrator-managers, between five and ten systems students, two or three operators, and an administrative assistant. The working model that developed, which we refer to as the small group model, has three main areas that apply to systems administration: the training of systems administrators, daily management (or daily survival), and how we get along with our customers. For the most part, the policies and procedures from the small group model are still in use.

Training

UnixOps does very little formal training of new administrators. This is in part because it's difficult to allot large chunks of a staff member's time to prepare and present classes. Another part of the problem is that system administration is not something that can be easily taught in a classroom environment – the problems are too varied and the answers seem to change constantly.

The main thrust of our training is familiarizing a new employee with the mechanics of the job – answering the phone, using Rand's MH and the Trouble MH [1] system for responding to customer requests – modifying campus databases and forwarding phone messages. We also ask that they read a beginning systems administration book – usually Nemeth, et al. [2]. These books don't often help the system administrator deal with day-to-day problems, but they provide the background necessary to understand the solutions presented to them by current staff.

Further training is provided in two ways – the first by occasional seminars presented by a manager or senior student, and the second by one-to-one instruction. The latter takes on many forms. Often we present new administrators with a problem we believe they are capable of handling, let them figure out as much as they can on their own, and then assist them out of dead ends, or confirm the correctness of the solution they have chosen.

Occasionally a senior staff member will walk through a particularly complicated process, such as installing C News, with a junior administrator. For the most part, training consists of making junior admins confident enough to tackle day-to-day problems, and answering any questions they come up with. Sometimes the answer is "have you read the man page?" However, one of the few UnixOps policies is that all questions from staff members take priority. The response may be just a pointer to where the answer is found, but there must be a clear, courteous response.

Survival Techniques

Many elements help our day-to-day operations run smoothly. This does not mean that things always run well, just that things would be a lot more chaotic if we didn't have these tools and policies.

Our central tool is the trouble queue – this is the Trouble MH system which has been in use at UnixOps for at least four years. This system allows students to control their own workload, and allows the staff to monitor progress on problems and offer corrections or advice as necessary. It also provides an audit trail, and to some degree, a knowledge base, as all request messages and subsequent responses are logged. Any status report or resolution on a trouble mail message is mailed out to the trouble mail readers. This keeps current work in front of all staff members, allowing them to better answer customer questions, and providing a measure of quality control – if one person makes a mistake, there are ten people who might catch it.

Occasionally a problem will arise with the "catching" of mistakes – one student's message highlighting the mistake of another may appear to be more of a personal attack than a professional criticism, regardless of the intention of the sender. However, these problems are usually worked out quickly, either by a face-to-face discussion between the two parties, or by a temporary moratorium on "flaming"[1] in a trouble mail mistake message.

Our most important policy, that of giving priority to questions asked by employees, has already been mentioned. This does not mean that a customer should be interrupted, but that non-life-or-death work should be put down to answer the question of another staff member. This may mean that senior students and managers often get interrupted, but in the long run more work gets done. The question will have to be answered eventually, and if it is answered immediately, the junior administrator won't sit around idle. This policy also keeps junior employees from being frustrated because they have nothing to do, having reached the limit of their knowledge.

Another implied policy is that of keeping the office home-like. This means keeping a microwave and refrigerator around, along with occasionally bringing in food for everyone (Elizabeth Zwicky mentioned chocolate chip cookies [3] – we find that muffins and bagels tend to keep the administrators from sticking to the walls. We do break down and have donuts every once in a while). It means realizing that school, in the long run, is more important than work, and being flexible to accommodate that. It also means allowing students to use the workstations for schoolwork when they aren't doing office hours, and allowing them to install software needed for schoolwork on work machines. The office is on a card access system, so that students may use the office at any time.

[1]flaming: term defined by USENET to mean excessive or gratuitous abuse.

A refrigerator and a microwave aren't that impressive – many workplaces have such amenities. However, the concepts of using work resources for school and bending schedules to accommodate school are somewhat unusual. The latter is fairly easy to justify – the goal of the university is education, and it wouldn't make sense for our organization to get in the way of that. This is best practiced in moderation – a staff member who is signed up for 18 credit hours and continually bows out of his 20 hour/week office hour commitment needs to either cut back on his office hours or drop a class.

This sort of unrealistic scheduling has not been a problem. Students set their own work schedules at the beginning of the semester, including the total number of office hours they will commit to. It only takes eight students working ten hours per week to have two students in the office during normal business hours. If there are more than eight students, or if those eight students work more than ten hours/week, the week gets covered without the manager having to assign office hours to fill the gap. On the occasion that a gap does occur, there are still full-time staff around to cover it.

Our reasons for allowing students to use office workstations for school are fairly simple. Most of the CPU time that is being used for schoolwork is time that would otherwise be spent idle. There are times when work and school related tasks are occurring at the same time on one machine. However, the school related tasks usually do not add a noticeable load to the machine. It is conceivable that a student might slow down a machine during business hours with a large compilation, but this has not occurred. Part of what makes a good systems administrator is courtesy and a sense of responsibility – someone with these qualities should have sense enough to save the large CPU jobs until after hours, or perform them on a non-work machine. Mis-use of company workstations has not been a problem.

In fact, mis-use of the office in general has not been a problem. Part of the solution is that we think of our office as a learning environment, and encourage students to spend as much time there as they like. In a very real sense, the more time that someone spends on a UNIX workstation, the more she[2] is going to learn about the system – making her a better systems administrator.

Not only does a pleasant and useful office provide more resources for the education of the students working there, and increase their systems knowledge, but it often results in problems being solved after hours – work that we can not afford to pay someone to do. Someone who is working late at night on a project is probably going to be willing to answer the phone, or walk upstairs and spend half an hour fixing a broken card access system, instead of letting the problem sit until the morning. Someone who has stopped by the office to read news might get curious about that persistent ''huge ethernet packet'' message on the netlog[3] and spend time investigating. Someone who can just eat a bag of microwave popcorn instead of having to go home to eat will probably stay late enough to finish a large job that would have taken longer if done in parts. In general, these fringe benefits don't cost much, and the work and expanded knowledge base that is gained would cost hundreds of dollars if paid for by overtime.

Customer Relations

Machines being fixed during off hours doesn't hurt our customer service image, either. However, we do not depend on this, or the fact that happy employees usually make happy customers, to be the basis of our relations with our customers.

Unfortunately, we cannot always keep our customers happy by fulfilling their requests quickly and completely. To do so, we would need three times the staff, and three times the budget we have now. However, we can at least keep them informed – let them know when projects will be completed, how close they are to completion, and when machines will be down. To this aim we have set up a distributed message of the day (motd) system that fits in with our aliases, hosts, and networks distribution system. In this way we can keep all of our customers up to date on network outages and other campus-wide downtime. For some of our customers, being informed is all they need to be happy.

We try to keep individual customers informed by rapidly acknowledging the receipt of trouble mail messages, and by keeping our service definitions available via anonymous ftp. The latter includes our ''Localization Checklist'' – the list of network settings and software that we install on a machine to get it integrated with the campus network. This checklist also resides in the root directory of every machine which we set up, and a customer can read this checklist to see what has and has not been completed. We also maintain an alias on each machine that points to the technical and administrative contact for that machine, so that even if a staff member does not know who owns a machine, he can still inform the owner of any changes made to the system. Another alias, ''diary'', goes to a file on the machine to log important changes for future administrators.

[2]In an attempt to be politically correct and still use proper english, we have tried to make equitable, yet consistent, use of gender-specific pronouns.

[3]We have all machines that we administer report errors to a central host. This host allows anyone logged in as the user ''netlog'' to see these error messages as they arrive.

Finally, we have a set of pagers that rotate among student staff members. The number is listed in our voice-mail message for emergency use, such as a downed card access system or a machine on fire. However, the effectiveness of this system is limited by the fact that we cannot obligate our staff to appear on-site whenever they are paged – doing so would cost too much in overtime. While the staff member carrying the pager is obligated to make a phone call, the decision to go to the workplace is left entirely to the staff member carrying it. Yet most problems can be solved over the phone (especially the ones called in by fellow staff members). By rotating the pagers on a weekly basis, and keeping the numbers somewhat hard to find, (thus limiting the number of people "crying wolf"), most of the pages get resolved without burning out a particular employee. Most pages are made by junior staff members or operators who encounter problems they can't resolve.

Sometimes user relations also means dealing with uncomfortable situations – most notably the situation of being asked to fix something which was broken by a user with root access, as part of a full-service contract. This is somewhat analogous to restoring a file for a user who has accidentally deleted it, as opposed to restoring a file lost due to hardware problems. Thus our policies are also similar: we will restore for free a file lost due to "natural" causes, but charge per hour for a file deleted by a user. We will also fix, for free, a system that we "broke", or was otherwise not caused by a user with root access. Yet even for customers who have a full-service contract, we charge per hour to fix other people's messes. Fortunately, this policy has never been enforced.

Expanding the Scale of Small Group Administration

These policies and concepts worked quite well until about a year ago. On the whole, customers were reasonably pleased with our performance, and the staff felt that their jobs were important and enjoyable. However, after a complete changeover in management, the doubling of student staff drove the organization a lot closer to chaos. The two new managers became swamped with learning how things worked, and the students who still needed guidance got lost. The time it took to respond to customer requests started going up, and some difficult problems were put aside indefinitely.

Hence the need to expand upon the small group model – the main reason being to provide better support for the junior staff, who by now outnumber the managers and senior students combined. There are two primary goals for this reorganization: the first is to make the organization a cohesive unit, as it was before the expansion. The second goal is to train new administrators faster – to accelerate the normally quite slow process of bringing new administrators to the point where they can handle most problems without help. Our main intention is to expand the small group model, as opposed to replacing it altogether. This should keep the changes minimal enough to be easy to implement while still accomplishing something.

Additional Survival Techniques

The biggest change is the addition of another level of responsibility that formalizes the status of senior students as mentors: more accessible than the managers, and able to answer most questions. Each senior student is placed in charge of a small group of (three to four) junior students. The senior student becomes a team leader – someone who not only acts to organize and lead the group, but also takes a fair share of the work. This frees up time for the managers, provides leadership experience to the senior students, and increases dissemination of knowledge to the junior administrators. The team leader does not replace the managers, but is instead added to the junior administrator's quick list of people to turn to for help.

Taken as a unit, these groups are small enough to re-create the small group conditions. The team leader takes the responsibility of not only planning and working on the team projects, but also increasing the knowledge of her team members. Our belief is that systems administration cannot be learned in a classroom. Even small seminars only serve to give the attendees an introduction to the topic. Ninety percent of the knowledge about that topic comes from hands-on exploration. (with the manual or a knowledgeable party by his side) on the part of the learner. Thus a senior assigned to four junior administrators should serve to increase the amount of time a team member can be learning directly (or indirectly) from another person. We also assume that, as in most tutoring situations, the tutor will receive some knowledge from the tutored.

The team leader not only learns from her team members, but also learns about her team members, and is thus able to help her team members choose tasks that are challenging, but not so difficult that the junior administrator will either fail to complete it, or require hand-holding the whole way through. This should also speed up the junior administrator learning process.

The second aspect of our improvement program involves extending the "staff empowerment" aspects of small group administration. The premise is that if a student is responsible enough to be a systems administrator, she is also responsible enough to select tasks that she will be able to complete, and assign her own office hours. Large group administration also assumes that the student knows enough about systems administration to know what qualities, aside from prior knowledge, make a good systems administrator – and thus we have made staff interviews part of our hiring process.

We hope that this will benefit us in two ways. First, staff interviews will allow current staff to determine how likely the interviewee is to fit in well with the organization. If a manager is still undecided about a particular applicant, the impressions of current staff can be very important. The second foreseeable benefit is that, if hired, the applicant will take less time to fit in and to feel comfortable interacting with the rest of the employees.

Another aspect of staff empowerment is the creation of ''standing orders'' – policy statements which give student administrators permission to bypass normal policies in an emergency. For instance, we cannot add a user to a machine without the permission of the owner of that machine. However, one of the standing orders is that in an emergency (for example, if the machine was advertising bad routes and disrupting traffic on the campus backbone) an administrator may add himself to the machine with root capabilities in order to log in and fix the problem. Standing orders rely on the discretion and responsibility of the student administrators, and enable them to cut through political ''red tape'' when absolutely necessary.

The final new ''survival tactic'' is the idea of the ''three Cs'' for dealing with a crisis situation: stay cool, calm, and collected. Most of our junior administrators can handle things after only a few months of training. After that, it is only a matter of making them realize it. We try to stress the idea that no problem will be helped by panic.

These ideas are not particularly new or innovative – the innovation is their application to a small university group staffed almost entirely by students. It's easy for students to look at this job as just a student job to earn money. However, our aim is to make this organization as professional as possible – to come as close to ''real world'' standards as we can with university funding and a staff that averages only fifteen work hours per week.

Better Customer Relations

A large part of being professional is treating your customers well. This is very easy to forget in a university environment, where most services have only one provider. This is almost, but not quite, the situation we occupy, and so we forget that the ''users'' are also the customers. For instance, about six months ago a professor called to ask about the X window system, and when the student on the other end put the phone down the professor heard the student refer to him as a moron.[4] Our improvements to customer relations are intended to remind the staff that UnixOps exists to serve the customers, and that customers are not annoying faceless entities. Another goal is to remind and/or teach the staff that not everyone is as comfortable with computers as they are. The obvious first step is to increase direct interaction with the customers. When a customer calls asking about an X question, a student should walk up the customer's office, instead of trying to debug the problem over the phone. The resulting personal contact changes the interaction from administrator-and-user to one person helping another. The same change is accomplished by increasing the number of office hours that student administrators hold in the workstation labs, and dividing the hours up among several administrators.

[4]This particular anecdote might be seen as a contradiction of the idea of the responsible student administrator. We feel that one rude remark does not necessarily indicate that a student is irresponsible – though that particular student certainly needs to work on his tact.

The second improvement once again relies on student responsibility. The general idea is to eliminate the manager's role as intermediary between someone who wants work done and the administrator who actually does the work. If a client asks a manager to upgrade her machine, the manager will assign a student or team to the task, and then have the student or team leader contact the client for the specific details. This should result in faster response on customer requests, and reduce the load on the management.

Another improvement to customer response time should result from a particular student or group taking responsibility for handing requests and problem reports from a few specific clients. If a student is already familiar with that customer's specific machine or lab, the customer does not have to continually re-explain unique setup or software, and the student should be able to find the problem faster. This is actually something that happened informally several years ago, when we had fewer customers and a larger senior staff. It worked very well then. However, this direct contact would be lost after the UnixOps employee graduated, leaving the customer back to explaining everything again. Yet, if an entire team is aware of the general setup of all the team member's contact machines, the loss of knowledge that occurs every spring should be significantly reduced.[5]

Results of Large-scale Improvements

Training and Survival Techniques

These ideas have been slowly phased in over the last six months. The last idea that we implemented was that of permanent sub-groups, about a

[5]There is one potential downside to this practice. If a customer only contacts one administrator, that customer will have trouble getting a response when his contact is busy with other work or school. However, this problem is easily resolved if the contact simply forwards the problem on to the trouble queue or notifies the client of the her temporary unavailability.

month ago. However, many large projects over the summer were handled by temporary sub-teams. The temporary teams worked largely as expected. They eased the load on management and got the job done faster due to teamwork. The teams did not work perfectly, as one team leader completely monopolized her project, leaving no work for her team members. However, one of the recently created sub-teams managed to upgrade a two-machine office in three days, involving all three team members. The upgrade went as planned – which is saying a lot in terms of the plans of systems administrators.

Possibly the most successful aspect of these changes has been the combination of paying more attention to selecting tasks for junior administrators (as carried out by the managers, as opposed to team leaders) and the idea of staying cool, calm, and collected. Three students – two who were hired at the beginning of the summer and one who moved up from doing backups at about the same time – have blossomed over the summer. They are now at the point where they can work independently, needing little or no help even with obscure problems. One of these students, with minimal supervision, installed four SMD drives and a color board on a Sun 3, two months after starting work.

Student staff participation in the interviewing process has been less successful. At the beginning of the summer five new systems administrators were hired as a result of a month-long hiring process which included staff interviews of all the applicants. Those hired integrated with the rest of the staff much faster than did previous new hires. However, the ease that these new staff members felt may have hurt as much as it helped. Often, the unease felt by a new employee in his new job will prompt him to work hard at learning and performing his occupation. Without this unease, two of the new hires spent a lot of time doing very little, and another spent a lot of time on the phone with personal calls. The former problem has been solved by putting the two less-motivated individuals in a sub-group whose leader is one of the managers, who has focused on keeping them busy. The latter problem seems to have abated with the start of school.

In general, things have improved, now that the sub-groups have been formed. Staff members are not getting lost in the shuffle and problem reports are receiving faster replies. Our general response time is still not as good as it could be. And while the new staff have fit in well, we are still not operating as a cohesive entity.

Customer Relations

Our customers also seem to be happier. The increased lab time is working very well – over the summer two undergraduate labs were almost completely upgraded, and now each lab has a server running BSD-based UNIX (SunOS 4.1.3) with several clients running System V based UNIX (Solaris 2.2). Given the potential for confusion between the two operating systems, we have not had any complaints, and the users seem to be handling the change with a minimum of difficulty, thanks to the in-lab advisors. Staff members are leaving the office more often to fix printers or check out problem reports, speeding up the resolution of those problems.

Conclusions

While things are a lot less chaotic around the office these days, it is still too soon to judge the overall success of the large group model. Subgroups seem to be the most promising innovation, but there is still one pitfall to overcome: group dynamics. At least at this university, undergraduates are not taught how to work well in groups. The only classes that require group effort are senior-level, final project courses. As such, we may have to establish our own curriculum for promoting teamwork, or find a suitable course outside the university.

Involving current staff in the interviewing process is another promising idea, but it has not worked as well as we hoped. Our first results have proved to be somewhat mediocre, and the schedule juggling required to achieve those results leaves the system, as it now stands, too costly and counterproductive to use. With the juggling to schedule an interview with the managers and the wait for a time when at least three staff members were in the office, the whole process took too long. It took so long that one good candidate dropped out of the running.

We have not abandoned the idea completely, however. The hiring process needs to be streamlined. Possibilities include forming an interview committee to handle the entire interviewing process, or pre-selecting a group of students to represent the student staff, and arranging a regular time when this group can meet with prospective employees.

For now, we have gone back to the manager-only interviews, as we are still expanding our staff. The third professional (non-student) manager started work a month ago, and we recently hired two operators and an administrative assistant. The five student administrators who started at the beginning of the summer are, as a group, only now becoming confident and effective. It may be six more months before the people who make up our organization feel comfortable working with one another.

The temporary sub-groups and the training of our new administrators over the summer have generated more observations that may prove useful in further expanding the large-group model. Assigning a new employee to a mentor from the current staff, at least for a week or two, brings the new employee up to speed much faster than the aforementioned book-and-project method. We are also thinking about a "localization checklist" for new employees

– a checklist of things for a mentor to teach his novice how to use: MH, voice mail, pager, microwave, stereo, etc.

We want to instill a sense of pride in doing the job, and in doing it right the first time. In the past, UnixOps has had a reputation for effective and robust solutions. Many administrators who graduated from this organization have gone into system administration jobs in the "real world" and continued this tradition. Our ultimate goal is not only to continue to provide the best service possible, but to give students the skills necessary to succeed in any professional endeavor to which they apply themselves.

Acknowledgments

We would like to acknowledge Bob Coggeshall and Lynda McGinley, who (knowingly or not) created the small group model.

Author Information

Tim Hunter is in his fifth year of the Electrical and Computer Engineering program at the University of Colorado at Boulder. He is in his fourth year as a student administrator at UnixOps. He recently spent the summer working in the "real world" at Odyssey Research Associates in Ithaca, NY. He can be reached via electronic mail at tim@colorado.edu.

Scott Watanabe is a Systems Manager at UnixOps at the University of Colorado at Boulder, where he also received his B.A. in micro-biology. He provides the students at UnixOps with moral support, technical guidance, and chocolate pudding. He can be reached at watanabe@colorado.edu.

Both authors can be reached at the University of Colorado, UnixOps/CNS, Campus Box 455, Boulder, Colorado 80309-0455.

Bibliography

[1] T. Hein, E. Nemeth, and T. Galyean, "Trouble-MH: A Work-Queue Management Package for a >3 Ring Circus", USENIX LISA Fall 1990 Proceedings.

[2] Nemeth, Evi, Garth Snyder, Scott Seebass, *UNIX System Administration Handbook*, Prentice Hall, Englewood Cliffs, 1989.

[3] "System Administration Tools Your Vendor Never Told You About: The Chocolate Chip Cookie", Elizabeth Zwicky, *;login:*, Vol. 18, No. 1, p. 10.

Role-based System Administration or Who, What, Where, and How

Dinah McNutt – Tivoli Systems

ABSTRACT

Traditionally, access for performing system administration tasks is an all or nothing proposition. With root access, an administrator can potentially make many changes to a system even though you may only want to allow them to add a user or mount a filesystem. In addition to specific tasks, you may want to control what tasks an administrator can perform based on which machine they are using. For some tasks, you also want to manage how those tasks are performed. For instance, when you add a user, you usually want to make sure the user ID is unique and is not zero.

This paper defines requirements for a role-based system administration environment. It describes and compares traditional solutions such as restricted shells, multiple root accounts, and setuid programs. The comparisons are made in the context of the requirements defined and are used to introduce the motivation and need for an alternative solution.

The solution proposed in this paper is object oriented and is based on the draft POSIX 1003.7 standard. Where appropriate, specific implementations (such as the Tivoli Management Environment) will be referenced. These examples will include lessons learned at Tivoli in developing and using an object-oriented system administration tool.

Introduction

UNIX treats almost all resources as files: disk drives, terminals, and executables. A standard file access mechanism is used to determine which process has what type of access to other resources (e.g., files, other processes, devices, etc.) This mechanism uses owner, group, and world permissions to allow read, write or execute access. On some UNIX systems, access control lists are supported which allow you to set exceptions to the standard file access for individual users or groups of users.

Most system administration functions such as adding new users, adding new devices, and editing system files, must be done as the root user. If a user can become the root user (by either using the su command or logging directly onto the system as root), the user can by-pass the traditional UNIX file protections. This is a security problem because the user can now have access to files they wouldn't normally be able to access. Traditionally, this problem has been treated as an ethical one and good system administrators do not go around and reading files that belong to other users unless it is necessary to troubleshoot a problem. As UNIX gains popularity in commercial environments, data security requirements may dictate that good ethics are not enough to solve the problem.

In a production environment, logistical concerns loom even larger. Why should I have to give someone the root password just so they can create users or mount a filesystem? Let's look at some of the issues and define some requirements for system administration tools to help solve these problems.

Requirements

The requirement for a system administration environment are summarized by the following:

- *Accountability* - Who is logged on as root? If everyone uses the root account to do system administration tasks, it is difficult to know which administrator is logged onto the system.
- *Auditing* - Who did what and when? This requirement supplements accountability by telling you what the administrators are doing.
- *Defining task-based roles* - Ability to allow different administrators to do different tasks. I may want "Joe" to be able to add users and do system backups, but I want "Mary" to be able to reboot the system and add users. This is a mechanism above and beyond the all or nothing capabilities of the root account.
- *Automation* - Automating redundant tasks should be simple. You can write scripts or program to automate tasks, but sometimes writing the tools is more effort than doing the task. A good system administration tool should make this task easy.
- *Distributed solution* - Don't repeat the same task on each of your 100+ systems. You should be able to execute one command (or series of commands) and have changes take effect on all systems.
- *Extensibility* - You should be able to customize vendor software if needed. Vendors

should supply as complete a solution as possible, but every site is different and you may have some exceptions the vendor did not anticipate.

- *Heterogeneous* - Most sites are not homogeneous and it would be ideal to have a system administration tool that ran on every system you have. Operations and commands should work transparently across systems.

Traditional Solutions

Multiple root accounts

Some sites control who can log in as root by creating multiple accounts with a UID of 0. Here is an example of multiple accounts using a UID of 0:

```
kernie:Tlxig45qKWShc:0:10:kernie:
     /home/kernie:/bin/csh
kiva:KlEW6bcwJan7s:0:10:kiva:
     /home/kiva:/bin/csh
otto:21WByNSB8jMPY:0:10:otto:
     /home/otto:/bin/csh
```

Each of these users probably has a different account they use for normal activities and these special accounts used for tasks that require root access. Since each of these root accounts is independent, once a user no longer needs the root access, the special root account can be removed.

The most obvious problem with this approach is that most UNIX applications use the account UID instead of the login name. Once a user logs in using a "second root" account, the user in indistinguishable from the "real root". For example:

```
rockytop% su kernie
Password:
rockytop# whoami
root
```

What has happened is that the (*whoami* command has taken the UID 0, looked it up in the password file, and returned the first entry in the password file with a UID of 0. This is also true when you edit files. The user's UID is stored in the inode of the file, not the username.

Multiple root accounts do allow you to assign private root passwords, so you can give out selected root access to machines, without disclosing the root password used everywhere (if you use NIS or NIS+ for distributing the root password). However, this approach provides little information about who is doing what, since all actions appear to have been performed by the "root" user.

Restricted Shells

Restricted shells allow you to define a restricted environment from which users can run a limited number of commands. The traditional interactive shells (*sh, csh, ksh,* etc.) allow the user to run potentially any command. A restricted shell allows you to define which users can run what commands.

There are several restricted shells available: *sudo, resh, rsh,* and *sush* are examples. In general, these shells share the following features:

- The ability to define which commands a user may execute on a per user basis.
- Logging of who executed what command.
- Password protection so that each user must enter their password when accessing the restricted shell.

Some shells are interactive where you type the name of the shell, then execute commands just like you would from the C-shell or Bourne shell. Other restricted shells require that you pre-fix the command you wish to execute with the name of the shell. For instance:

```
sudo /bin/tar cvf /dev/rst0
       /usr/local
```

You need to be careful to not give the users access to commands that could be used to by-pass the restricted environment (like *wftp* and *wvi*.) There is a special version of *wvi* called *wrvi* available that does not allow you to escape to a shell.

The advantages of restricted shells are the flexibility and accountability they provide. One disadvantage is that you must install and maintain the source code for the shell since a restricted shell is not standard on all UNIX systems. The exception is that *wrsh* is available on System V. Many large companies would prefer to pay a vendor to provide this functionality and free their staff for other tasks. Another disadvantage is that you must configure the users of the shell on each system you wish to manage. This could be a very time consuming process if each system has different administrators.

Captive Accounts

The biggest different between a restricted shell and a captive account is that a user can start a restricted shell from his or her own interactive account, but must log in to a captive account directly in order to gain access to the environment defined by the captive account.

Often, these shells must be executed by the root user. To use the shell, simply create a special account for the restricted shell user and give the account UID 0 as if you were creating a second root account, However, define the interactive shell in the password file to be a restricted shell, and you can limit what each root-privileged user can do. The same advantages and disadvantages for restricted shells apply for captive accounts.

Setuid programs

Setuid programs are set to mode 4755 or something similar. The setuid bit, 04000, is the important part. Users who execute these programs actually run as owner of the file for the duration of the program.

If you have a very specific task, you can write a C program or perl script that contains all the commands needed to do the task. Set the owner of the file to be root and the file mode 4755. Then any user can execute the file. You can also set the mode to 4750 and the group ownership to a specific group (e.g., *rootusrs*.) Then you can add any users you want to be able to run this script to the *rootusrs* group.

Setuid programs are useful to delegate specific tasks to a set of users as long as you have a manageable set of tasks. A complex site with many combinations of systems and pseudo-privileged users may not find this a practical solution for many tasks. In addition, be cautious with setuid scripts, since they are a common source of security holes.

An Object Oriented Solution

The basic problem we are trying to solve is "Add Joe User to these 50 systems."The fact that you have 10 HP/UX systems, 10 Suns, 20 AIX systems, 9 SGIs, and 1 NeXT is irrelevant. It would be nice to execute the *add_user* command and have all the right things happen to each system.

As a system administrator, you want to know what the *add_user* command is doing and be able to modify the procedures it calls to add this user. Once it is set up, you should to be able to run the command without worrying about what it is doing.

Under this scenario, as more types of systems are added to the network, you would only have to make minor changes to the *add_user* procedure on the new host.You wouldn't have to train your staff how to administer the system since they would already know how to use the system-independent system administration commands.

This solution has three basic components: a command-line interface, a programmatic interface, and a set of managed objects. Some specific applications may also include a GUI-based interface. The managed objects encapsulate the information required to administer the objects. In other words, the user object on an AIX system would contain the commands required to add a user to that system. The fact that these commands are different than those required to add a user to a Solaris system would be transparent.

This strategy is based on the POSIX 1003.7 standard and assumes that a management platform or framework that supports communication between the different systems and supports the implementations of a common set of managed objects (users, groups, printers, etc.) exists on all systems. This framework includes a common API to the framework which would provide access to the services provided by the platform, including the ability to access objects whose locations are unknown to the application. This framework is outside the scope of POSIX 1003.7 as it is being addressed by OSF, UI, ISO/OSI, COSE and others.

Managed Objects

The POSIX 1003.7 (or POSIX.7) standard attempts to define managed-object classes that are descriptions of both the properties (attributes) and behavior (methods) of logical and physical resources that are managed on a system or systems. Examples of managed-object classes include user, printer, and host. Properties of the user class might be login name or home directory. Behavior methods would perform tasks like creating the home directory or changing the user's password. This reference model

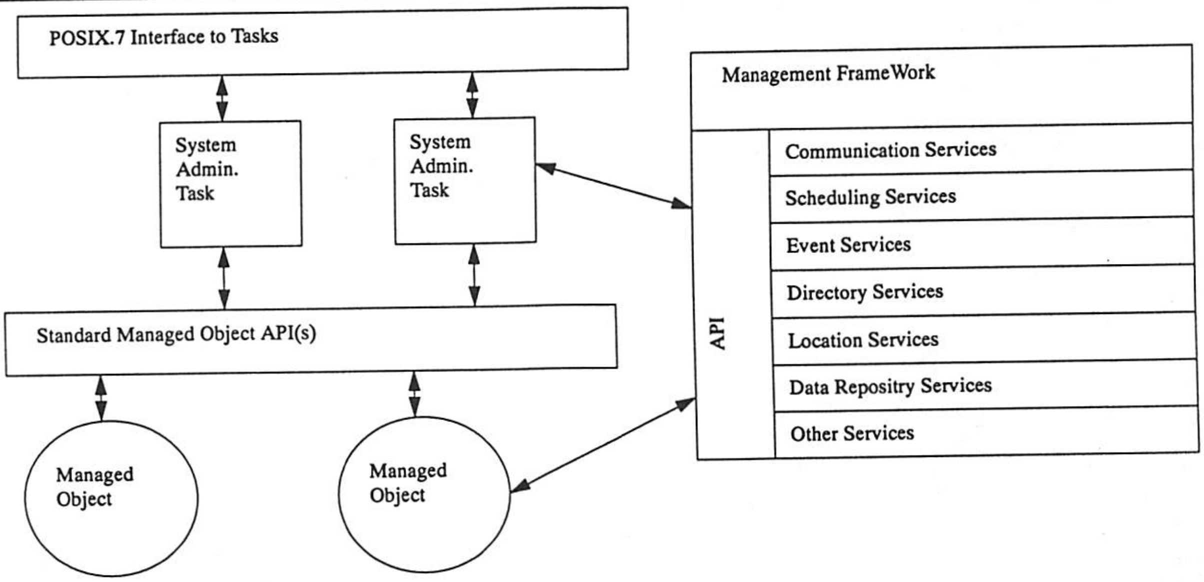

Figure 1: POSIX.7 Reference Model

is shown in Figure 1. Assumptions made by this model include:

- System administration is a task-oriented process.
- A task is a high-level operation, such as "export filesystem" or "delete user".
- There are three different interfaces: a command line interface (CLI), and applications programming interface (API), and a graphical user interface (GUI.) POSIX.7 only addresses the CLI and API.
- A task may internally invoke other tasks.
- The management framework provides common services such as naming/location, notification, data management, communication, authentication, and authorization.
- The location of the managed objects is transparent.
- Management operations are typically simple, policy-free, low-level atomic operations (such as changing a user's UID in the password file or disabling FTP service on a host.)

In this environment, applications work as sets of cooperating objects. These object must delegate authority to other objects in order for tasks to execute correctly. The delegation must happen in a secure manner to make sure someone is not able to perform a task they have not been authorized to perform. Role-based security is one way to implement a secure delegation mechanism.

Roles

As mentioned earlier, there is a need to perform system administration tasks without using the all or nothing capabilities of the root account. The management environment described in the previous section provides a mechanism for defining different roles for different administrators. Since all the information needed to manipulate an object is encapsulated within the object, you can extend the attributes of the object to include roles. A simple role-based model could be patterned after UNIX file system access: read/write,read-only, or none. If you have "read/write" access to an object, you can modify or delete the object by invoking the one or more of the object's behavior methods. "Read-only" access allows you to examine the attributes of an object and "none" would not allow you to view any information about the object.

The next step is to identify the system administrators to the management environment. Ideally, I want to be able to log onto a system as "dinah" and perform tasks under that user ID. The management system must be able to provide some kind of authentication that I really am "dinah" and determine what kind of authorization I have. This implies that authorization and roles must be pre-defined by one of the administrators. Let's look at the following matrix:

	Printers	Users	File Systems
Jamie	RW	RW	N
Larry	RO	N	RW
Shoe	N	RW	N

Using the Read/Write, Read-only, and None notation described earlier, you can define access for three different administrators with distinct, but overlapping job functions. If Jamie executes the *add_user* command, the management environment would determine what access was required to add a user by getting the information from the attribute on the user object. Next, it would look up what access Jamie has been authorized with respect to that task. The two must match before the task will be performed.

In reality, you want more complex operations than read/write, read-only, and none. By carrying this model one step further, we can define specific administration tasks. For instance, let's define the task "Manage Printers". This high-level task might include the following steps:

- Reset print queue
- Define new printer
- View print queue
- Delete job from queue
- Re-order print queue
- Delete printer
- View accounting log

For each sub-task, we assume there is one or methods that must be executed in order to perform the task. Each method may have an access control list (ACL) that defines type of access is required to execute the method. The Read/Write, Read-only, and None access is no longer sufficient. We must define task-based roles for different types of managed-object classes. Using "Manage Printer" as an example, we define the following roles:

- *Senior Administrator* – Reset print queue, Define new printer, View print queue, Delete job from queue, Re-order print queue, and Delete printer
- *Junior Administrator* – Reset print queue, View print queue, Delete job from queue, and Re-order print queue
- *Manager* – View accounting log
- *None* – No access

Some tasks (such as Reset print queue) may have more than one role associated with it. Our administrator task matrix now looks like:

	Printers	Users	File Systems
Jamie	Sr.	Sr.	None
Larry	Jr.	None	Sr.
Shoe	None	Sr.	None

Note how you can mix and match roles between administrators. The next step is to add locale. Keep in mind this is a distributed application, so you don't necessarily want to give someone the ability to manage all the printers on your network. It makes more sense to delegate administration of a subset of the printers to staff in the department that owns the printers or staff located physically near the printers. Now we expand the printer portion of the task matrix:

Who	What	Where
Jamie	Sr.	Bldg. 6
	Jr.	GLOBAL
Larry	Jr.	Bldg. 5
	None	GLOBAL
Shoe	Sr.	7th floor
	None	GLOBAL

The resources designated by the "Where" column are arbitrary. The administration software must give you the flexibility to define a collection of resources and designate them as "Bldg. 6" or "7th floor". You should not have to be bound by NIS domains or DNS sub-nets since those are very often not the same as the administrative boundaries. You may also want to group resources residing on the same computer system into one or more locales. The GLOBAL locale indicates the role of the administrator for all other locales besides the ones specified.

The last step is how these tasks are performed. For user management, you may want to create home directories in a different location depending on the locale of the user. The task matrix can be expanded to indicate how a task is to be performed:

Who	What	Where	How
Jamie	Sr.	Bldg. 6	/home/<username>
	Sr.	Bldg. 7	/user/home/<username>

The "How" is tied to the "Where", not the role of the administrator. This feature gives you the flexibility to define how a task is performed for a specific set of resources which can even exist on the same host.

Delegation and Authentication

There are many different ways this type of role-based administration system could be implemented. The important features to look for are flexibility and secure authentication. You want to make sure the system will make things easier for you and allow you to delegate tasks to less experienced system administrators. Most of us have created more user accounts than we care to admit and are delighted to turn that role over to someone else. Secure authentication (such as Kerberos) is important because you don't want other users to masquerade as the users who are authorized to perform system administration tasks.

By delegating some of the tasks through the roles we defined above, we aren't giving total control of the system to the different administrators since the root password is no longer required. Each administrator can have as big (or small) a role as you define and you are free to focus on more difficult tasks.

Real-life example

At Tivoli, I have gained experiencing in using a system administration application similar to the one describe in this paper. The Tivoli task matrix for creating users looks similar to:

Who	What	Where
Jamie	Sr.	7th floor
	Jr.	GLOBAL
Larry	Jr.	6th floor
	None	GLOBAL

Tivoli refers to the locales as policy regions which is an arbitrary grouping of resources your can manage with the Tivoli software. The Tivoli software defines 4 roles: Senior, Admin, User, and None which map into the 4 roles we defined earlier: Sr. Administrator, Jr. Administrator, Manager, and None. Administrators may have different roles in different policy regions and you can define what type of resources each policy region may manage (users, printers, etc.)

My experience has been that 4 roles is not enough. You need to be able to define an arbitrary number of roles depending on the task and the locale. At some locales, you may want to allow a certain user to reset a print queue, not not perform any of the other tasks associated with managing printers. Referring to the roles we defined for administering printers, the junior administrator was allowed to: Reset print queue, View print queue, Delete job from queue, and Re-order print queue. Ideally, you would like to be able to define the new role of "local administrator" who may only re-set the printer queue. Fortunately, this is not a hard problem as the ACL are already built into the objects. You can simply add a new role to the ACL. The only software changes required are to those portions of the software that allow you to define the roles themselves.

References

S. Garkinkel and G. Spafford, *Practical UNIX Security*, O'Reilly and Associates, 1991.

D. McNutt, "Role-based System Administration", *SuperUser*, May, 1993.

B. Lampson, "Authentication in Distributed Systems: Theory and Practice", *ACM Transactions on Computer Systems*, November 1992.

J. Steiner, et. al., "Kerberos: An Authentication Service for Open Network Systems", *Usenix Winter 1989 Conference Proceedings*.

T. Smith, "An Object-Oriented Approach to UNIX Systems Management", *SANS Conference Proceedings*, Spring, 1992.

Author Information

Dinah McNutt is a System Administration Consultant for Tivoli Systems where she works with customers of Tivoli helping them with system administration problems and customization of Tivoli's system administration software. She has been doing system administration for over 8 years and has written technical articles on the subject for SunExpert Magazine, RS/Magazine, and the X Resource Journal. Ms. McNutt currently writes the Daemons and Dragons column for UNIX Review magazine. She can be reached at dinah@tivoli.com.

Delegation: Uniformity in Heterogeneous Distributed Administration

Jean-Charles Grégoire – INRS-Télécommunications

ABSTRACT

We describe a distributed administration tool based on the concept of delegation. Management operations are lightweight processes created on a manager's machine, but executed remotely on specific targets. Various libraries are provided on target machines to cover the widest range of administration tasks such as modification of configuration files, recurrent, scheduled executions of programs or process monitoring. The result is a single, unified framework for the majority of administrative operations, and a distributed platform to facilitate its use in a flexible way. The language used to realize the platform is small, unobtrusive and robust.

Introduction

System administration has slowly evolved from the historical machine-based perspective to distributed environments. In the Unix universe, administration is still a blend of machine-based operations combined with distributed applications, typically based on a client-server model. The historical heterogeneity of interfaces of machine-based systems has increased in the process.

Tools are built to alleviate the administration burden. They however very seldom cover the whole spectrum of operations, focusing rather on a specific range of problems. They are applications, or programming languages and they contribute to the plethora of information an administrator has to deal with – a form of *cognitive overload*.

We introduce here a distributed management platform based on the concept of *delegation*[2]. Management operations are lightweight threads, downloaded from the manager's host machine to selected targets. Specific language features guarantee that the operation requested is meaningful. The first release of this platform is operational and exploited across the machines of our group at INRS-Télécommunications.

In this paper, we describe the requirements a distributed management platform has to meet, describe the tools typically used and their main features. We introduce the notion of delegation and show how it supports very naturally distributed administration. We present the language used to implement delegation, its salient features and the additions we made to it. We conclude with the discussion of some unresolved issues.

Management Activities

We distinguish between two different categories of administration operations: *static*, or slowly changing, and *dynamic*, or quickly changing.

Static administration

Static administration englobes the customization and tailoring of a system to a specific site's requirements, that is, system configuration. It is mainly done at system creation time, and updated when a change is required. The most frequent changes occur when user configuration or file system structures are modified.

Examples of configuration operations includes kernel modification, file system creation, and maintenance of a variety of files under the */etc* directory.

Most changes to static configurations are immediate, although kernel modifications require a reboot to be effective. Some applications may also have to be signaled, or restarted.

Dynamic administration

This form of administration is more focused on the dynamic behavior of the system: processes and their characteristics, user sessions or I/O performance in general. The relevant information is typically held in the kernel, and made available through system calls. In this case, it has the several consequences:

1. The information is potentially costly to acquire
2. The information is imprecise (e.g., ps listing)

Some dynamic information is also reflected directly in the file system, such as pending print jobs, or process information in some versions of Unix.

Distributed Administration Tools and Methods

As networks become more pervasive, a number of tools have appeared to support distributed administration. Some are proprietary commercial platforms, but many are now bundled with most operating systems. Tools like the Network Information Services[4] (NIS, formerly YP) or Hesiod[3] remove the management of file-based static configuration away from the individual system. Based on a client-server model, they provide a single point of update of the information, and one or more points of query, for optimization. These tools have typically focused on the distribution of static-type information.

When better mechanisms to support the creation of distributed applications became available, other tools have also been built to resolve specific static or dynamic problems: improved distributed print management systems (e.g., ISPIN), distributed batch queues (e.g., qbatch, CONDOR) or software distribution update mechanisms (e.g., DEPOT).

New commercial offerings such as Tivoli Systems' Management Environment[9] go further by providing an extendible platform supporting remote operations across an heterogeneous system. At this stage though, such tools are still heavy to use, resource hungry, and often cumbersome to manipulate. They are typically aimed at very large networks.

For services not supported by distributed applications, the administrator is left with the usual practice of establishing a connection to the machine he wants to work on, and use his favorite tools, such as **perl**[6] or the ubiquitous **sh/awk/sed** mix to perform the required operation. A recurrent operation will be captured in a program, to be executed on specific request, or at regular intervals using the **cron** mechanism.

Remote connections as superuser however raise some security concerns in systems where passwords are transmitted un-encrypted, or through the use of mechanisms such as **equiv** files.

The bottom line for the administrator is a mix of machine-based and distributed tools with different programming interfaces, overlapping but incomplete functionalities, or simply new, but potentially easier ways of doing the Same Old Thing.

We offer to this state of affairs our perspective of a *distributed, uniform, programmable platform* for system administration.

Delegate

We have developed a *delegate* approach to administration, with a *unifying* perspective on the various activities we have described above.

A delegate has the following features:

1. Uniform language interface across all platforms,
2. One language for all activities,
3. Interface to most static and dynamic configuration information,
4. Built-in scheduling.

We also have the following overall goals in focus:

1. Localization of management information,
2. Local information processing,
3. Fine grained configuration,
4. Reliability through sound semantics.

Whereas the current trend in distributed services consists of moving static information away from the workstation, we take the opposite perspective. Since the facilities to support static configuration already exist on all machines, and the operating system still supports them, we make use of them. Rather, we remotely control the updates of this information. All queries are local, rather than remote as in the distributed model. Interestingly, this makes the system more robust, since it is isolated from server failure,[1] but also allows the fine tuning of the configuration of the systems, unlike the "all the same under the same domain" approach of some distributed administration tools.

Dynamic information is acquired only when requested by delegated activities, and shared between threads. It is processed locally and algorithmic operations can be performed as required. For example, processes consuming too many resources can be re-niced, or killed. This is the reverse of the SNMP view of transferring data to a manager, which makes the decision and orders an action from afar. With delegation, responsiveness and reliability are improved.

Implementation

The delegate model has been implemented by extending a functional language with some specific features. There are several reasons for this decision. We did not want to design yet another programming language, and thus chose to start with an existing base. **Caml-Light** was chosen over more Unix-flavored languages such as **Perl** or **TCL**, because of its inherent simplicity, the simplicity of its implementation, its two-tiered implementation model, based on a bytecode, and the existence of well-defined semantics. We do not imply that more mainstream languages could not satisfy our requirements but only that **Caml-Light** was the most manageable starting point.

Caml-Light is a *pragmatic* functional language, i.e., with a limited notion of side effects,

[1] although only to the extent that the working files are held locally

derived from the ML language, developed at INRIA. It is strongly typed and modular. We have extended the basic implementation to support remote execution. More specifically, we have added:

1. Multiple threads of execution,
2. Cooperative and preemptive thread scheduling,
3. A runtime command language for thread management,
4. Dynamic library loading,
5. In-memory binaries management,
6. Inter-threads communications.

The runtime environment was modified to operate from a socket connection, rather than interactively. The garbage collector was also extended to support multi-threading.

Threads are compiled into bytecode on the administrator's machine, then transferred to the specific target where they are linked and stored. Interface definition files hold the type and the syntax of the operations available on the required targets to reduce the probability of runtime errors. Our goal is to make threads completely safe.

The runtime environment supports thread management. Threads' runtime execution is controlled and they are executed sequentially.

Ideally, a thread implements a single activity, such as the update of a file or the enforcement of some *policy*[7].

Support Required

Most support for administrative operations is provided by platform-specific libraries, with a generic interface. Libraries currently in use include a kernel access library and a user administration library. An interface to a SNMP library is in the works.

No change to the kernel was required although access to more precise information would obviously help in problem tracing.

Security

Security is, understandably, a major administration concern. Problems are worse in a distributed environment than on a single machine, as confidential information is transmitted, sometimes in un-encrypted form, across the network. Other concerns, such as spoofing, also emerge.

The typical answer to this problem is to make sure that only encrypted information circulates, and only authorized users manipulate the information.

Our delegate platform exploits the basic Unix protection mechanisms. The runtime listens to a protected port, and accepts sessions only from a corresponding privileged port.

This however provides only the typical ''all-or-nothing'' style protection. Only the superuser can use the communication, unless the management program is setuid'd, which defeats the whole purpose.

Caml-Light provides us with another interesting protection mechanism. This language uses the module concept for strongly typed modular compilation of programs. We can use the mechanism to provide different interfaces to different people. Protection hinges thus around two mechanisms: protecting the access to the management platform and restricting the operations that can be performed, and restricting the management programs which can be created by the different users.

We feel that we have enough latitude to implement secure access schemes, without unduly restricting access to the tool.

Let us note that the different library interfaces scheme is a quite interesting way to implement administration delegation. Combined with the Unix file protection system, it is possible to statically define which functionality any user would have access too. Any user, with access to the sources can also define a proper subset of the functions for other users, simply be writing an adequate interface.

Administrator Environment

The administrator has a simple environment to compose threads, compile and download them and control their activation and execution on remote targets.

Threads can send information back to the administrator machine through a log channel. There is no specific demultiplexing protocol at this stage. Threads must identify themselves, their host together with the data transmitted to allow demultiplexing and potential dynamic visualization on the administrator's host.

At this stage, a simple combination of **grep**ing and **tail**ing is enough to extract and present the relevant information.

Example

The program in Appendix A is an example of recurrent file system monitor. Each 45 seconds, all entries in /etc/fstab are processed, and the local file systems are examined. Two functions, *search_char* and *search_line*, provide general line scanning support.

This code is provided as an illustration of concepts and of **Caml-Light** and therefore does not attempt to be optimal. Ideally, getfsent should be used for portability and a list of the file systems should be maintained. The support routines should be in a general library, in scanf compatible form, although **Caml-Light** provides a much more flexible mechanism which is not demonstrated here.

Conclusions

We have described the features of a distributed administration platform based on delegation. We are currently experimenting with a first release of the system. Our experience so far shows the flexibility and the power of the delegation paradigm, mainly for dynamic administration.

It does not however cover all aspects of administration at this stage, and we need, for example, a special mechanism to transmit user-requested operations, such as a password change, to the administrator's site. We plan on expanding our model to integrate such operations in the context of *delegation of authority*. We also need to improve the quality of information threads can send to the administrator's environment.

The conviviality and the features of the administrator's environment will also receive further attention, to facilitate the administration of a large number of machines.

Author Information

Jean-Charles Grégoire is assistant professor at INRS-Télécommunications where he is involved in research activities in distributed systems and network management. He can be reached via snail-mail at INRS-Télécommunications, 16, pl. du Commerce, Ile des Soeurs, Verdun, Qc, CANADA, H3E 1H6. He can be reached electronically at gregoire@inrs-telecom.uquebec.ca

Bibliography

[1] X. Leroy, M. Mauny, "The **Caml-Light** System, Release 0.5, Documentation and User's Manual", INRIA, Sept. 1992.

[2] J.-Ch. Grégoire, "Management with Delegation", IFIP '93: AIP Techniques for LAN and WAN Management, April 1993.

[3] S. P. Dyer, "Hesiod", Usenix Conference Proceedings, Winter 1988, pp. 183-190.

[4] SUN Microsystems Inc., "Network Information Services".

[5] T. Christiansen, "Op: a flexible tool for restricted SU access", Proceedings of LISA III, 1989.

[6] L. Wall, R.L. Schwartz, "Programming Perl", O'Reilly and Associates, 1991.

[7] E. D. Zwicky, S. Simmons and R.Dalton, "Policy as a System Administration Tool", Proceedings of LISA IV, pp. 115-123.

[8] Network Working Group RFC 1157 "A Simple Network Management Protocol", May 1990.

[9] Tivoli Systems, "Tivoli Management Environment", 1992.

Appendix A: Sample Caml-Light Code

```
(*****************************************************)
(*   Identify Local File Systems with High Usage    *)
(*****************************************************)
#open "NEW_unix";;         (* unix function calls *)
#open "NEW_timing";;       (* recurrence mechanism *)
exception FS_NOT_LOCAL;;  (* file system not local *)
exception Found of int;;
exception Next_loop;;
(* Find position of a character within a line.      *)
(*        Return value = position in line            *)
let search_char line c =
 try
  for i=0 to ((string_length line)-1) do
   if (nth_char line i) == c then raise (Found i)
  done;
  -1
 with Found i -> i
;;
(* Extract string between the first pair of ':' *)
let sub_line line =
 let pos_1 = search_char line ':' in
  if pos_1 != -1 then
   let pos_2 =
    search_char (sub_string line (pos_1+1)
       ((string_length line)-pos_1-1)) ':' in
    if pos_2 != -1 then
     sub_string line (pos_1+1) pos_2
    else
```

```
      sub_string line (pos_1+1)
       ((string_length line)-pos_1-1)
     else
        ""
;;
let process_fs fs =
 try
  let data = statfs fs in
    (* calculate percentage utilization *)
    let t = float_of_int data.fd_btot in
     let f = float_of_int data.fd_bfreen in
      let u = t -. f -. (t *. 0.1) in
       (* Maximum utilization set to 80% *)
       if (u >. 80.0) then (
        print_string
  ("\nhostname: system getting full: "^fs^"\n");
  print_endline s;
       )
 with sys__Sys_error s ->
  prerr_string ("\nhostname:statfs: "^fs^"0);
  print_endline s
;;
(* Extract file system names from "/etc/fstab" *)
(*       and check utilization                 *)
let check_local_file_system() =
 try
  let in_fstab = open_in "/etc/fstab" in
   while true do
    try
     let in_line = input_line in_fstab in
      (* Only local file system names *)
       if (search_char in_line '@') == -1 then (
        (* Extract name and check utilization *)
        let fs = sub_line in_line in
         process_fs fs
       )
    with End_of_file -> close_in in_fstab;
     raise Next_loop
   done
 with sys__Sys_error s->print_endline s;exit 1; ()
    | Next_loop       ->flush std_err; ()
;;
(* Every 45 s., check utilization of file system *)
while true do
  print_string "\nhostname:Checking file system.\n";
  check_local_file_system();
  print_string "\nhostname:Done.\n"; flush std_out;
  delay 45
done
;;
```

satool – A System Administrator's Cockpit, An Implementation

Todd Miller, Christopher Stirlen, Evi Nemeth
– University of Colorado, Boulder

ABSTRACT

Monitoring a large number of machines in a distributed environment can be time consuming and inefficient. Often a system administrator discovers there is a problem with a machine or network link when a frustrated user calls. **satool** provides a way to efficiently monitor groups of machines and identify problems and potential problems quickly; it's sort of an early warning system for sysadmins.

satool is composed of three independent parts: an SNMP (Simple Network Management Protocol) agent that runs on each machine being monitored, a database that collects data from each client machine, and a graphical user interface (GUI) that acts as the interface between the user and the database.

Introduction

With the advent of fast, inexpensive workstations, the standard computing environment has changed from a few Vaxen to a distributed network that has become increasingly complex and difficult to administer. Monitoring large numbers of machines in a distributed environment can be time consuming and inefficient. In such an environment, it is usually not possible to see the warning signs that point to imminent disaster for a machine or network. Instead, system administrators find themselves fighting fires when their time would be better spent elsewhere. **satool** provides a way to efficiently monitor groups of machines and find problems and potential problems quickly in a straightforward manner.

The **satool** server maintains a database of client machines and the values of supported variables gathered during the last poll. Normally, a client is added to the database when the server receives a message from the SNMP trap daemon that the client's SNMP agent is running. However, the server also maintains an on-disk copy of the database that is read in on invocation of the server. Because of this, the database will survive system crashes. Client machines can also be specified in the server's configuration file to "prime the cache" of machines to monitor. The time between polls is configurable on a per-machine basis (overriding a stated default). The server listens for database requests from the display system on port 0xFB1 (that's a one not an I).

The **satool** display system includes an X based GUI (Graphical User Interface) built using **tcl/tk.** It supports a hierarchical top level view of the machines being monitored. Machines can be grouped to allow the screen area to scale with the number of machines being watched. For example, a sysadmin can configure his workstation to display the machines he is directly responsible for, the machines that his colleague on vacation is responsible for, and his network gateway to the outside world. After traversing the hierarchy to an individual machine, the display contains visual widgets representing the health of that machine: disk space free, memory statistics, cpu activity, mail queue length, nfs statistics, etc. The user can choose the form of the widget (currently text, a thermometer display, or a sliding window histogram) to display each quantity.

Alarm conditions for each variable can be expressed in the configuration files as well. If the value of a variable crosses the alarm threshold, a special action (blinking, beeping, reverse video, digital pager, etc.) is taken on the screen to indicate it. Alarms are propagated to the top level display, thus if /var/tmp fills up on host *heineken* on the *beers* subnet of the *cs* domain and if the groups are set appropriately, the alarm condition would be noted in the widgets representing *heineken, beers,* and the *cs* domain. A user would see or hear the alarm independent of which level he was actively displaying at the time.

A working prototype of **satool** exists. It will be used this fall by the Computer Science Department's undergrad lab sysadmin group and graduate/faculty research sysadmin group to monitor about 300 machines. We expect to use feedback from these groups to expand the sysadmin MIB and to build more display widgets. The code will be freely available with a Berkeley style copyright notice.

Design Goals

satool was designed to be *flexible*, *scalable*, *easy to manage*, and to *reuse* existing tools.

Flexibility is achieved through the use of configuration files and **satool's** modular design. On the data gathering side, configurable items include: the actual data to be collected and how often to collect it. From within the display system the user can also configure threshold values that indicate a problem and the action to take if a threshold is exceeded. Display system configuration also selects the hosts or devices to display, the type of widget to represent a variable, and personal preferences for items like the arrival of an alarm condition (for example color, sound, blinking, etc).

We wanted **satool** to be used on both small and large networks, thus *scalability* was important. Data collection has a low impact on the host and the networks on which **satool** is running. Allowing the user to interact with hosts in a hierarchical manner also increases scalability.

When dealing with large numbers of hosts *manageability* is essential. **satool** provides sensible defaults for parameters in configuration files. No changes are required to monitor a new host; it will be added when it sends a trap to the server. It is important to note that if the correct site-specific values are compiled into **satool,** there is only one configuration file to maintain, the server's.

In designing **satool** we also wanted to *reuse* tools where possible to avoid reinventing the wheel. To this end **satool** takes advantage of many standard UNIX utilities to gather data. We also use the **tcl** and **tk** languages from John Ousterhout at the University of California, Berkeley for the display and SNMP 1.1b from Carnegie Mellon University. An SNMP agent and sysadmin MIB (Management Information Base) provide for communication between the data collection and data gathering activities. The actual data gathering is done by an SNMP agent running on the hosts being monitored. We have extended the CMU SNMP 1.1b agent and MIB to include variables of interest to UNIX system administrators not included in the standard network or host MIBs.

Components

satool is made up of three distinct components: a data gathering agent, a data collecting server, and a display system. The agent is an SNMP agent extended to use the **satool** MIB. It uses "helper scripts" to massage data from standard UNIX commands. The server polls agents at set intervals and stores the resulting data in an *ndbm*(3) database. It also services requests from the display system using an SMTP-like protocol. The display system (written in **tcl** and **tk**) interacts with the user and initiates data transfers from the server.

satool Daemon (satoold)

satoold has three main functions: gather data from its clients, store it in a database, and service requests to access that data from the GUI. It also writes its process id to a file (/etc/satoold.pid by default) for convenience.

Data Gathering

satoold polls clients for data via SNMP at configurable intervals. All polling is done by a forked process which passes data back to the parent via a UNIX domain socket. Polling times for clients are kept in a linked list (called the "timer queue" although it is not truly a queue) sorted by time to poll (in UNIX time format). To allow concurrent polls, a compiler-time variable specifies the number of simultaneous polling processes allowed. A counter keeps track of the number of polling children along with an array of their process ids.
The flow of control is as follows:

- **satoold** is notified by the SNMP trap daemon that a client has come up.
- The client is inserted into the timer queue if it is not already present there.
- If the client is not already in the database, it is added. Otherwise, the client's current database entry is updated to reflect the fact that the client is now up.
- An alarm goes off, signifying that it is time to poll a client.
- A signal handler is called, and a child is forked to poll the client.
- The child completes the poll and sends the data back to its parent via a UNIX domain socket.
- If the connection to the client times out (the timeout is defined at compile time), that machine is now marked as down and its polling frequency is reduced (however, the polling frequency never reaches zero).

Data Storage

Client data is stored in an *ndbm*(3) database, keyed on the fully qualified hostname. The use of *ndbm*(3) allows for some basic crash recovery since a copy of the database is kept on disk. As such it can survive system downtime.

- The on-disk database is read on invocation of **satoold** and a timer queue is created based on the information in the database.
- Obviously bogus (empty) keys are discarded (this is the most common cause of database corruption we have seen).

Servicing GUI Requests

satoold accepts connections on port 4017 (0xFB1 in hex). Connections timeout after five minutes of inactivity (configurable at compile time). The protocol used is similar to *SMTP* (Simple Mail Transport Protocol). The protocol commands are:

`HELO`	Say hello to the daemon;
`HELP`	Prints a short help message;
`LIST`	Lists the all clients in the database;
`GET`	Gets the data for a particular machine.

satoold responds to each command with a three digit completion code and a status/error message (in text) before returning the requested data (also like *sendmail).* The breakdown of the three digits is as follows:

First Digit:
  2 Command completed
  5 Command failed with a fatal error
Second Digit:
  0 Syntax
  1 Information
  2 Connection
  3 Host
  5 Data
Third Digit:
  The third digit differentiates between codes that have the same first two digits. For instance, code 220 is the "connection established" greeting, and 221 is the "connection closed" message.

satoold Configuration File

Parameters to **satoold** can be configured through its configuration file (/usr/local/etc/satoold.conf by default). The polling interval for most hosts is specified by the *interval default* entry. It defaults to 300 seconds if not specified. Individual host values specified using *interval hostname* entries override the default. Hosts to preload automatically, without receiving a trap from the host, are specified with a *preload hostname* entry. Anticipated use of the preload feature is to include main servers in the config file to prime the cache. By using host-specific intervals one could also poll those hosts more frequently.

A sample satoold.conf file follows:

```
# example satoold.conf
#
# interval default seconds
# interval hostname seconds
# preload hostname
#

interval default 300
preload hazelrah.cs.colorado.edu
preload alta.cs.colorado.edu
preload kinglear-gw.cs.colorado.edu
#preload fiver.cs.colorado.edu
interval romeo.cs.colorado.edu 600
interval alta.cs.colorado.edu 200
interval pipkin.cs.colorado.edu 400
```

SNMP Agent

The SNMP (Simple Network Management Protocol) agent used in **satool** is based on snmp1.1b from Carnegie-Mellon University. This release is MIB-I compliant. There are two major differences between CMU and **satool** versions: the configuration file and support for **satool** variables in the MIB (Management Information Base).

Config File

The **satool** SNMP agent's configuration file (/usr/local/etc/satool-agent.conf by default) is currently only used to specify the host to send coldstart traps to (in the absence of a configuration file a default host is used that is specified at compile time). On invocation, the agent will send a coldstart trap to the host listed in the config file (if that file exists) to announce its presence. The SNMP agent runs on the hosts being monitored and does the actual data gathering. We have extended the CMU SNMP 1.1b agent and MIB to include variables of interest to UNIX system administrators not included in the standard network or host MIBs.

satool Variables

The agent now supports variables defined by the **satool** MIB. The values for most of the variables are obtained via "helper scripts" that run standard UNIX commands and parse the output into a form that the agent can use. Their are two major reasons to use these "helper scripts." First is the increased portability and flexibility scripts provide. Second is our desire to use existing UNIX tools where available.

There is, however, a problem with the approach described above. It is extremely slow for a large number of variables because the agent does a *popen*(3) call which forks and execs the script for each variable. A solution is to have the script output all the variables we might be interested in at once and cache the values. For example, instead of calling a *vmstat*(1) helper script eighteen times, we call it once and cache the values for ten seconds (configurable at compile time). Subsequent requests for any of the variables will get the cached values. This speeds things up considerably and the ten second granularity is considered acceptable. The full **satool** MIB can be found in appendix A of this paper.

Sample MIB Entry

The following MIB entry describes the load average to an SNMP agent.

```
satoolLoadAve OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 1 }
```

The first line defines an object called satoolLoadAve. It is of type INTEGER (SNMP doesn't have floats so we multiply the load average by 100 and then truncate it). The ACCESS line indicates that satoolLoadAve is read-only (wouldn't we all like to see writable load averages!). The status of the object is "optional" because it is not part of the standard MIB. The last line describes where the object "fits in" to the MIB hierarchy (which is visually a tree). In this case satoolLoadAve is the first leaf in the satool branch. Its full path is .iso.org.dod.internet.private.enterprises.cu.satool. satoolLoadAve.0. The terminating zero indicates that satoolLoadAve is a leaf node.

Display System

The display system provides an X-based Graphical User Interface (GUI) for viewing the contents of the **satool** database. The GUI, written with **Tcl** and **Tk,** uses configuration files to setup **satool's** hierarchical view of the hosts being monitored, to specify thresholds for data values, and to set the type of display objects. The display system also checks for data values out of bounds and notifies the user when an alarm threshold has been crossed. It is intended to be run on a sysadmin's workstation or dedicated management station sitting quietly in the background until an alarm is triggered when it will notify the sysadmin of impending trouble.

Tcl and Tk

Tcl (pronounced "tickle"), which stands for "tool command language", is an interpretive programming language built from a library of C procedures. **Tk** is an X11 toolkit that is accessible from **Tcl.** They were developed by John K. Ousterhout at the University of California, Berkeley.

Tcl/Tk's strengths are its portability, extensibility, and communication. **Tcl** and **Tk** have been ported to most UNIX based platforms that support X11R4 or higher. Scripts written in **Tcl/Tk** can be extended and modified at run time without recompilation. **Tcl/Tk** provides a powerful communication command called *send,* which allows different **Tcl/Tk** processes to communicate with each other.

Tcl/Tk was chosen for the **satool** project (over **Interviews** or **Suit**) because it is mature and easy to use. Chris, who did the GUI part of **satool,** took the manual home one evening for bedtime reading. By afternoon the following day, after only four hours playing with it, he had a small application built. After this experience other windowing toolkits were not seriously considered.

Configuring the Display

satool represents groups of hosts hierarchically, very much like the netgroup concept defined by Sun. The groups can be nested; there is only a practical limit to the depth of the hierarchy. Clicking on the icon representing a group zooms you to the members of that group.

Groups are defined by the configuration file satool-display.conf using the syntax of the /etc/aliases file, namely:

```
groupname: member,member, ...
```

A member can be either an individual host or a group, for example:

```
ugradlab: kinglear, kinglear_clients
kinglear_clients: hamlet, ophelia, \
       juliet, caesar, romeo
```

The alarm thresholds for each variable monitored are also specified in this file on a per host basis. Macros are supported, so that a group of machines with the same values can be configured in a single line. The syntax is:

```
macro = variable value, variable ...
machine: variable value, variable ...
machine: macro
groupname: macro
```

For example, to configure all the kinglear clients to use the same alarm threshold values:

```
hp_common = load_avg 5,num_procs 100
kinglear_clients: hp_common
kinglear: load_avg 10,num_procs 200
```

If a machine object is not mentioned in the configuration file, it will be assigned default values which are set in the GUI code. There are two additional entries: the domain designation and the server designation. Specifying a domain allows all hostnames following it to use short names, rather than fully qualified names. It is in effect until another domain entry occurs. The syntax is:

```
domain: domain-name
```

The server entry specifies the hostname of the machine that contains the database, for example:

```
server: kinglear
```

Display Objects

satool's display objects, currently a sliding histogram, a thermometer, and a text object, are used to view the data in the database. The frequency at which a display widget is updated can be changed by selecting a new value from the Frequency menu at the lower left corner of the window. The user cannot choose an update rate that is more frequent than the database polling rate. The default value for the update frequency is 5 minutes; it is set in the satool-display.conf file but can be overridden by the user.

We call a histogram which displays the values from the last *n* (10 by default) time intervals a sliding histogram. This type of widget is appropriate for variables like the load average and gives a sense not only of the current value but of its first derivative. The minimum, maximum, and running average are also shown. The histogram will scale as appropriate.

The thermometer widget is used to display one dimensional data. The current implementation uses a percentage value instead of scaling the data. The thermometer displays the current data value, the maximum and minimum running values, and the running average. Figure 1 shows a typical thermometer.

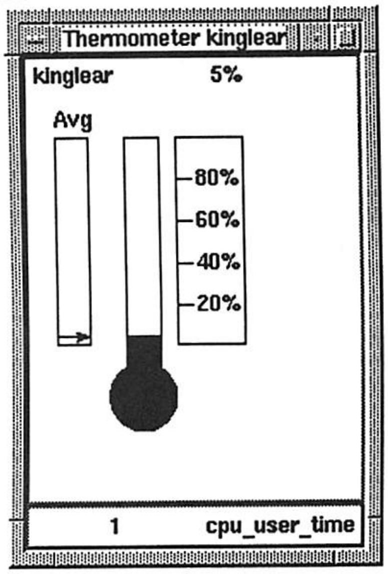

Figure 1: Thermometer

satool's text object displays the name of the machine being monitored and the requested data items. Four data values: average, minimum, maximum and threshold are shown.

Configuration

The sysadmin can configure their **satool** display to select the hosts whose data will be displayed, types of widgets to use, alarm thresholds and actions, and screen layout. The users configuration file is called .satoolrc and should be in his home directory. Examples of each configuration primative are listed below:

- configure display object for each data item

```
display: load_avg H, \
    disk_usage T
```

- configure alert colors and icons

```
alert1: green client1
alert2: yellow client2
alert3: red client3
alert4: black client4
```

(above syntax: alert_name: color icon)

- select groups from satool-display.conf

```
groups: kinglear_clients, alta_clients
```

- select macros from satool-display.conf

```
kinglear_clients: hp_common
```

- set thresholds

```
alta_clients: load_ave 3
```

- define groups

```
my_group: kinglear_clients, kinglear
```

- save state of display objects

```
Histogram kinglear.cs.colorado.edu \
    load_ave +20+20
```

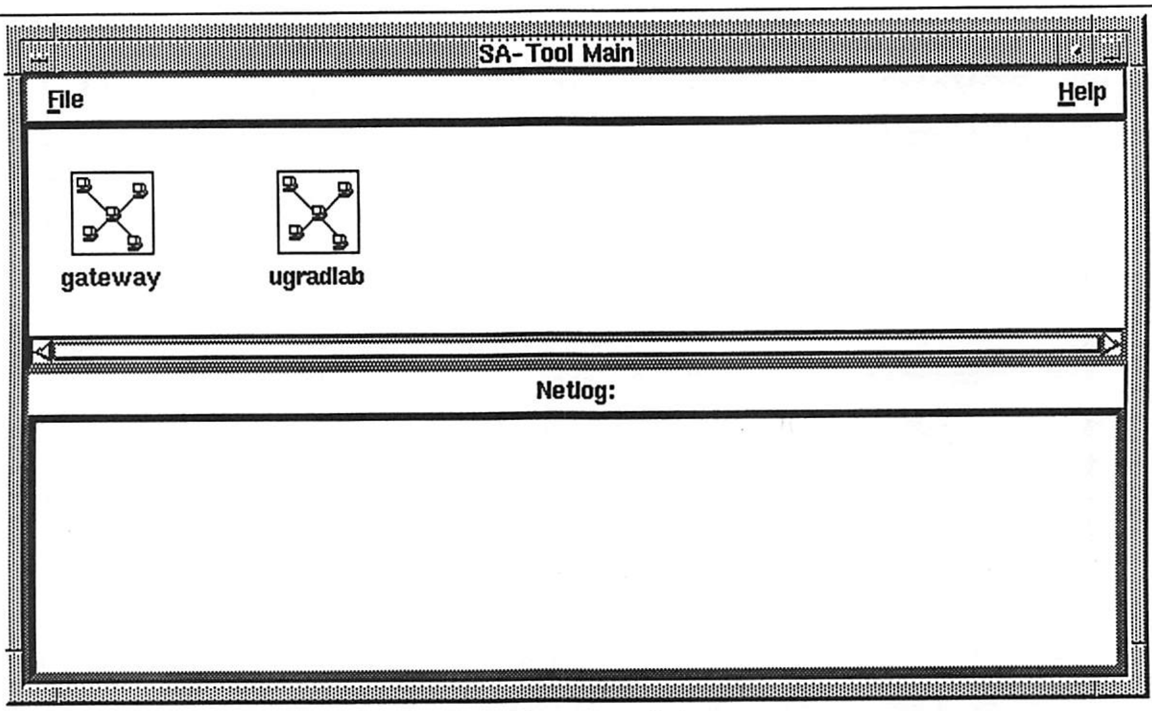

Figure 2: Main Window

- save state of main window

```
Main +274+245
```

- set default display object

```
default: X
```

- set frequency default (in minutes)

```
frequency: 10
```

- set number of histogram bars

```
h_bars: 15
```

- set pager phone number (also in satool-display.conf)

```
pager: 303-555-1212
```

Windows and Menus

In addition to the data display objects, **satool** has four other windows: a main window, viewer windows, help and message windows.

The main window contains the root of the host hierarchy. From it you can traverse the hierarchy via viewer windows. It also provides an interface to netlog, a local tool for displaying system events sent via the 4.3 BSD *syslog*(3) facility. Figure 2 shows a typical main window.

The main window's File menu has three options: Exit, Save Config, and Save Config on Exit. Saving the configuration writes the current screen layout, display options, and window locations to the users .satoolrc startup file. To configure a netlog section of the main window include the following line in your .satoolrc file:

```
netlog: loghost.domain
```

for example:

```
netlog: kinglear.cs.colorado.edu
```

Viewer windows show group membership and give the user access to the data **satool** monitors. The Data menu, which is opened by selecting a host, lists the data items available. The option All will display a text object showing all the data items for the selected host. The viewer window's Display menu allows the user to override the type of display object that will appear. Once the user has selected an option from this menu, it will remain selected until the window is closed or another option is chosen. Figure 3 shows a typical viewer window.

Help on the use of the GUI is available through the Help Menu.

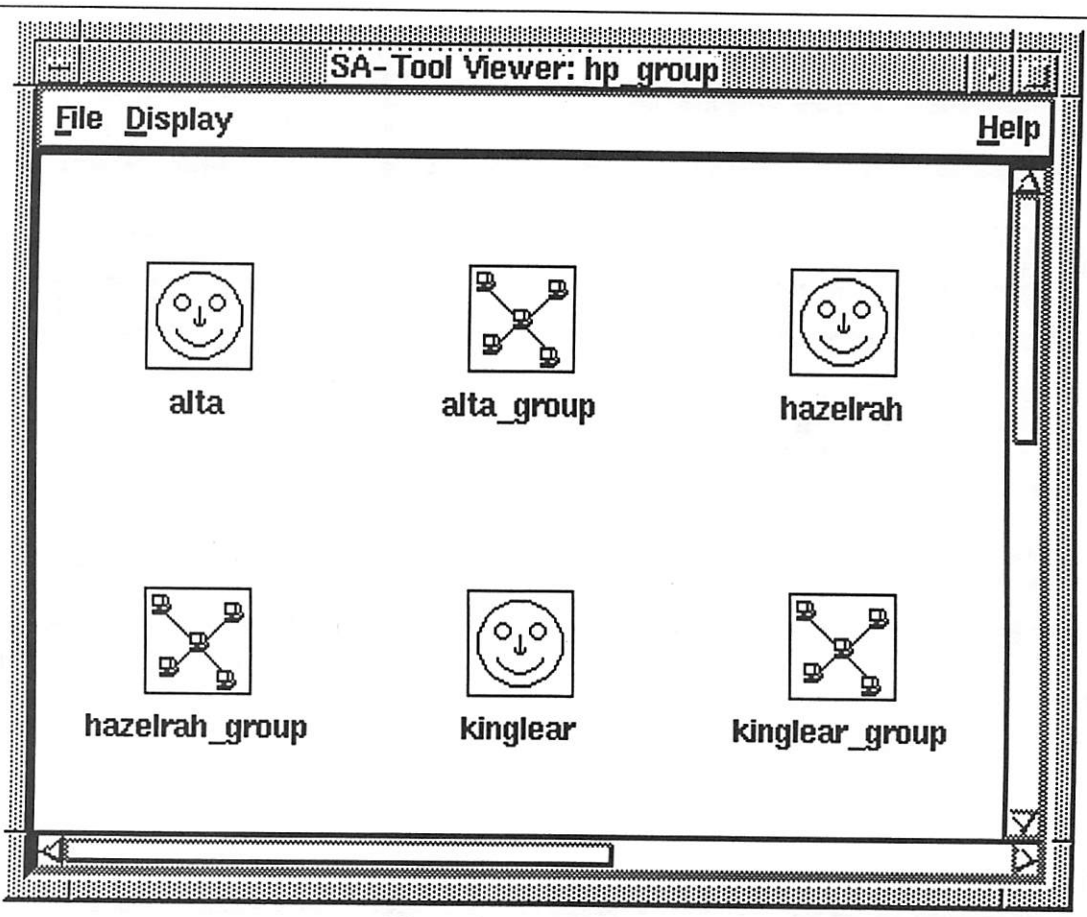

Figure 3: Viewer window

satool's alert system notifies the user when data values cross stated thresholds. The data object causing the alert will change appearance (color, reverse video, new icon, etc.); the alert will cascade up through the hierarchy.

In addition, the user can configure additional notification methods on a per machine basis. The current list of alarms implemented are:

- alert_alarm: sends annoying beeps
- alert_mtf: moves the offending object to the front of the screen
- alert flash: flashes the offending object
- alert_mesg: creates the message window
- alert_pager: page a human being

The message window receiving an alert will report the host, data item, current value and threshold value. It also displays error messages including configuration errors, start-up errors and errors in accessing the database.

Extending satool

Adding a new variable to monitor is fairly easy. Following is a list of the steps we took to include a new variable that does a *traceroute*(8) to an outside machine to make sure gatewaying is working correctly. In each case, the area where code needs to be added is marked with EXTEND_HERE in a comment.

Agent

The first step is to write a helper script that the SNMP agent will call. In this case, it is a *perl*(1) script called traceroute-helper. The script simply does a *traceroute*(8) to the machine enss.ucar.edu and prints 1 if it was successful and 0 if there was a problem (routing loop or host unreachable). This script must be placed in a directory in the agents path. The next step is to add the new variable to the MIB.

```
satoolGatewayOk OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 11 }
```

It is now necessary to modify the SNMP agent to run traceroute-helper when appropriate and update **satoold** to use the new variable. There are three places to change in the agent. The first step is to assign the new variable a NUMBER in the satool header file. This is used to identify the variable internally. The last number used is 48 so we can do:

```
#define SATOOLGATEWAYOK  49
```

Next, we need to tell the agent about the variable's existence. This is done in snmp_vars.c in the variables[ ] array. We just need to add a line like the following:

```
{{ENTERPRISES, CU_ENTERPRISE,
SATOOL_NUM, 11, 0}, 10, INTEGER,
SATOOLGATEWAYOK, RONLY,
var_satool }
```

after the last **satool** variable. Between the first set of braces is the numeric representation of the variable name, separated by commas. ENTERPRISES represents iso.org.dod.internet.private.enterprises and CU_ENTERPRISE is the enterprise number assigned to the University of Colorado. SATOOL_NUM is **satool's** number in the cu hierarchy, and 11 is our place in the **satool** hierarchy (the eleventh branch). The zero is used as a terminator. The "10" is a count of the elements in the first set of braces. INTEGER denotes the type of the data and SATOOLGATEWAYOK is our variable number. RONLY signifies that this variables is read-only. var_satool is the function that is to be called to resolve the variable. Now that the agent knows about the variable and how to resolve it, we need to write a resolution function. In this case, we just put the new variable in an existing function, var_satool( ). This function is in satool.c and currently resolves the load average and mail queue length. To add our new variable we just add the following code to the switch statement in var_satool( ).

```
case SATOOLGATEWAYOK:
fildes = popen("traceroute-helper",
     "r");
if (!fildes)  return(NULL);
if (fscanf(fildes, "%d", &n) < 0)
{
     pclose(fildes);
     return(NULL);
}
long_return = (long)n;
pclose(fildes);
return (u_char *) &long_return;
```

All we are really doing here is running our helper script and passing what it prints back as an int.

Server

We still need to tell **satoold** to monitor and store this variable. To do this we need to update three files. The first step is to add the new variable to the satool_variables struct in satool_db.h. We can add something like:

```
int gateway_ok;
```

Next we need to have this updated when the server polls its clients. The place to do this is in update_data( ) in load_values.c. We need to add two lines at the end of the if statements.

```
else if (!strcmp("satoolGatewayOk.0",
     name))
     sat_var_ptr -> gateway_ok =
       snmp_var_to_int(buf, variable,
       subtree->enums);
```

This converts the snmp variable into an int and stores it in the correct field of the struct. The next step is to add code to print the variable and value when the display system requests a host's data. We just need to add one line in the get_command() function in parse_client_request.c.

```
sprintf((char *)ret + strlen(ret),
     "%s %d ", "gateway_ok",
     sat_var -> gateway_ok);
```

This just adds a "variable value" pair to the return buffer. That's all it takes. If you want to add variables that are outside the **satool** part of the MIB you will need to add extra code to update_data(). This is not necessary from within the **satool** hierarchy because the SNMP getnext operator is used to get the variables (as such the server need not know them by name when requesting them).

GUI

The final step is to add the new variable to the GUI. It must be added to the default list in procedure BuildList, file IO.tcl:

```
gateway_ok 0 X
```

The variable name is followed by a threshold value and its default display object. A threshold value of 0 for a boolean variable triggers the alarm when the gateway is down; a text display object is specified. Note that the text display is the default and so would not actually need to be listed.

The new variable must also be added to the Data menu. This is done in the mkDataMenu procedure in Menu.tcl. The new option can be set as follows:

```
$parent.menu.data.misc add command \
     -label "Gateway Ok?" \
     -command "RunDisplay gateway_ok"
```

Since tcl/tk is interpreted there is no need to recompile; just restart **satool** and the gateway_ok data object will be immediately available.

Supported Architectures

Currently, the SNMP agent is only known to work under 4.3 BSD. The **satool** elements of the agent are known to also work under Ultrix 4.2/4.3 and should be easily portable to most versions of UNIX. The **satool** GUI will run on any system capable of running **tcl/tk. satoold** should run on any version of UNIX that supports Berkeley sockets and *ndbm*(3).

Future Enhancements

satool is currently deployed in the Computer Science Department's research and instructional networks. As experience is gained in this "pseudo-real-world" environment, we expect to continue improving it. Current plans include:

- MIB-II support
- Support for more architectures in the SNMP agent
- Extra modules to alert sysadmins of pending and existing problems (email and pager).
- Extra widgets for the display system
- More flexible alarms, including special handling based on the time of day and a muting option.
- Scalable alert thresholds including support for a bottom threshold as well as the upper limit.

Author Information

Todd Miller is a recent graduate of the University of Colorado, Boulder where he received a Bachelors Degree in Computer Science and served as a systems administrator for the last two years he spent there. You can reach him electronically at millert@alumni.cs.colorado.edu.

Christopher Stirlen is a masters student in Computer Science at the University of Colorado, Boulder. He currently works for XVT Software in Boulder as a Software Test Engineer. Chris has a strong interest in User Interface design and visual programming. Reach him electronically at stirlen@cs.colorado.edu.

Evi Nemeth is an Associate Professor of Computer Science at the University of Colorado, Boulder. Reach her electronically at evi@cs.colorado.edu.

References

Case, et al., *RFC 1067*, DDN Network Information Center, SRI International, 1988.

Hardy, Darren and Morreale, Herb, *buzzerd: Automated Systems Monitoring with Notification in a Network Environment*, LISA VI Conference Proceedings, 1992.

Nemeth, Evi, *SA-Tool, A System Administrator's Cockpit*, AUUG Conference Proceedings, 1991.

Ousterhout, John K., *An Introduction To Tcl and Tk*, 1993, available via anonymous ftp on sprite.berkeley.edu:tcl/bookps.Z.

Rose, Marshall, *The Simple Book*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1991.

Rose, Marshall and McCloghrie, Keith, *RFC 1065*, DDN Network Information Center, SRI International, 1988.

Rose, Marshall and McCloghrie, Keith, *RFC 1066*, DDN Network Information Center, SRI International, 1988.

Appendix A: satool MIB

```
-- sa tool
cu OBJECT IDENTIFIER
    ::= { enterprises 632 }
satool OBJECT IDENTIFIER ::= { cu 1 }
-- to get real load average divide
-- the int by 100
satoolLoadAve OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 1 }
-- number of entries in the mail queue
satoolMailQueueLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 2 }
-- number of system calls / sec.
satoolNumSysCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 3 }
-- process information
processes OBJECT IDENTIFIER
    ::= { satool 4 }
-- number of processes
satoolNumProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 1 }
-- number of processes in disk wait
satoolNumWaitingProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 2 }
-- number of zombied processes
satoolNumZombieProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 3 }
-- number of processes in the run queue
satoolRunQueueLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 4 }
-- number of processes blocked for
-- resources
satoolNumBlockedProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 5 }
-- number of processes runnable
-- but swapped
satoolNumRunnableButSwapped OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 6 }
-- vm information
vm OBJECT IDENTIFIER ::= { satool 5 }
-- number of context switches / sec
satoolNumContextSwitches OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 1 }
-- number of active virtual pages
satoolActivePages OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 2 }
-- number of free virtual pages
satoolFreePages OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 3 }
-- number of page reclaims / sec
satoolPageReclaims OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 4 }
-- number of pages attached / sec
satoolPagesAttached OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 5 }
-- number of pages paged in / sec
satoolPageIns OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 6 }
-- number of pages paged out / sec
satoolPageOuts OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 7 }
-- number of pages freed / sec
satoolPagesFreed OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 8 }
-- anticipated short term memory
-- shortfall
satoolMemLow OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 9 }
-- number of pages scanned clock
-- algorithm / sec
satoolPagesScanned OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 10 }
-- io stuff
io OBJECT IDENTIFIER ::= { satool 6 }
```

```
-- number of device interrupts / sec.
satoolNumInterupts OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { io 1 }

-- rpc stuff
rpc OBJECT IDENTIFIER ::= { satool 7 }

-- number of rpc calls (server)
satoolServerRpcCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 1 }

-- number of bad rpc calls (server)
satoolServerRpcBadCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 2 }

-- number of empty rpc calls (server)
satoolServerRpcNullRecv OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 3 }

-- number of rpc calls with too small
-- a body (server)
satoolServerRpcBadLen OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 4 }

-- number of rpc calls that failed to
-- decode into xdr (server)
satoolServerRpcXdrCall OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 5 }

-- number of rpc calls (client)
satoolClientRpcCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 6 }

-- number of bad rpc calls (client)
satoolClientRpcBadCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 7 }

-- number of retransmitted rpc calls
-- (client)
satoolClientRpcRetrans OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 8 }

-- number of rpc calls where the reply
-- transaction ID did not match the
-- request transaction ID (client)
satoolClientRpcBadXid OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 9 }

-- number of rpc calls that timed out
-- (client)
satoolClientRpcTimeOut OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 10 }

-- number of times the client had
-- to sleep
satoolClientRpcWait OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { rpc 11 }

-- nfs stuff
nfs OBJECT IDENTIFIER ::= { satool 8 }

-- number of nfs calls (server)
satoolServerNfsCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { nfs 1 }

-- number of bad nfs calls (server)
satoolServerNfsBadCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { nfs 2 }

-- number of nfs calls (client)
satoolClientNfsCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { nfs 3 }

-- number of bad nfs calls (client)
satoolClientNfsBadCalls OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { nfs 4 }

-- number times a client structure
-- was successfully gotten
satoolClientNfsNclGet OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { nfs 5 }

-- number times all client structures
-- were busy
satoolClientNfsNclSleep OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { nfs 6 }

-- cpu stuff
cpu OBJECT IDENTIFIER ::= { satool 9 }

-- percent of cpu in user time
satoolCpuUserTime OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
     ::= { cpu 1 }

-- percent of cpu in system time
satoolCpuSysTime OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS optional
```

```
    ::= { cpu 2 }

-- percent of cpu in idle time
satoolCpuIdleTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 3 }

-- percent of cpu spent running niced
-- processes
satoolCpuNiceTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 4 }

-- table from df
satoolDfTable OBJECT-TYPE
      SYNTAX  SEQUENCE OF DfEntry
      ACCESS  read-write
      STATUS  optional
      ::= { satool 10 }

dfEntry OBJECT-TYPE
      SYNTAX  DfEntry
      ACCESS  read-write
      STATUS  optional
      ::= { satoolDfTable 1 }

DfEntry ::= SEQUENCE {
  dfIndex
      INTEGER,
  dfDevice
      OCTET STRING,
  dfMountPoint
      OCTET STRING,
  dfTotalKb
      INTEGER,
  dfUsedKb
      INTEGER,
  dfAvailKb
      INTEGER,
  dfCapacity
      INTEGER
}

dfIndex OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 1 }

dfDevice OBJECT-TYPE
      SYNTAX  OCTET STRING
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 2 }

dfMountPoint OBJECT-TYPE
      SYNTAX  OCTET STRING
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 3 }

dfTotalKb OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 4 }

dfUsedKb OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 5 }

dfAvailKb OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 6 }

dfCapacity OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 7 }
```

Sysctl: A Distributed System Control Package

Salvatore DeSimone & Christine Lombardi – Project Agora, IBM T.J. Watson Research Center

ABSTRACT

The sysctl package is an authenticated client/server system for executing remote commands. It is conceptually similar to *rsh*, but adds Kerberos[1] authentication, an ACL-based command authorization mechanism, and a programmable Tcl-based[2] command language in its server.

The sysctl server component, **sysctld**, is a daemon that runs on all workstations. The client component lets users send sysctl commands to a **sysctld** server. If the user is authorized for the requested operation, the server executes it on behalf of the user and sends back the result. The operations sent by the user are processed at the server using a built-in command interpreter and can range from a single sysctl command to a complex sysctl script. The server has a multi-level authorization scheme to guard against unauthorized access to commands.

The sysctl server uses the embeddable command language Extended Tcl[3] as the foundation for its built-in interpreter. The server can dynamically link in external shell commands and Tcl procedures to integrate existing management tools or create new global or service-specific commands. Once a command is created inside a server's interpreter, it is accessible to any authorized user from any workstation.

Sysctl uses the Kerberos authentication service for reliable third-party authentication, a prerequisite for authorization checking in a distributed computing environment. The server's built-in authorization mechanism provides granularity down to the individual command level.

Introduction

The motivation for the sysctl package comes from the experience of managing hundreds of workstations. The Agora Project manages approximately 1000 workstations of various UNIX flavors at the IBM T.J. Watson Research Center. There are many software packages and services available to help in managing such an environment. At Agora, these include Kerberos[1] authentication, the Hesiod[4] name service, NFS and AFS file service, software maintenance programs such as SUP[5], and others. What is missing is a general purpose *glue* package that can integrate these pieces into a coherent, manageable system and fill the considerable gaps between them.

Problem Specification

Before discussing the details of the sysctl package and its operation, we present a list of the key problems it is designed to solve.

Security

The management tools used to administer a large site need to be secure not only to prevent outside intrusion, but also to prevent accidental disasters such as the inadvertent deletion of a critical data file. The standard UNIX security mechanisms were designed for a single system and are not sufficiently robust to provide security in a distributed computing environment. Kerberos provides a means of authenticating users in a networked environment, but the interface for using it is via C library routines. Most administrative scripts in Agora are written in perl[6], and there is little desire to re-write them in C to ''kerberize'' them.

Authorization

The term *authorization* is used here in a generic sense and refers to the ability of a central managing organization to assign a set of administrative privileges to a user. These privileges must be valid for a set of machines without giving that user administrative access to other parts of the environment. The inability to *partition* administrative privileges has been a problem at Agora. Many tools require *superuser* access on a particular machine, others use simple authorization methods based on host names or UNIX user names. Some systems, such as AFS, have advanced authorization schemes but do not provide the desired granularity. In most cases, the situation is *all-or-nothing*: a user has either no privileges or all privileges.

Specialized Services

A site that provides many services utilizes large numbers of servers with diverse management requirements. At Agora, we have found that frequently a separate script or daemon is created to maintain a particular service. For Agora, writing separate daemons from scratch is inefficient, especially when features such as Kerberos authentication require the code to be written in C. More importantly, writing separate daemons in the absence of an organized *methodology* has led to a splintering of the management tools. The interfaces to the servers are different, the methods for maintaining the servers are different, and authorization is usually omitted or done on an ad-hoc basis.

Location Independence

Many administrative tasks require the user to login to a particular machine to perform a function. In a distributed environment, the administrator should be able to issue commands from any workstation to any workstation.

Design Overview

The sysctl package consists of the following elements:

- Server program (**sysctld**) that runs on all workstations.
- Commands built-in to the server which form the base of the system control language.
- Configuration files that control aspects of the server operation as well as extend the command set available on a given machine.
- Client library (**libsysctl.a**) that contains a **sysctl()** function, providing a C language interface for communicating with **sysctld**.
- Client shell program (**sysctl**) which offers a command-line interface for communicating with **sysctld**.

Sysctl uses a client/server, Remote Procedure Call (RPC) model. The **sysctld** server is a multi-tasking server that runs on every workstation. Clients send commands to a server using either the client library routines or the shell command.

When the server receives a request, it uses Kerberos in combination with authorization lists to assign an *authorization level* to the client. The operation contained in the request is then passed to the built-in command interpreter for execution. The availability of built-in and external commands varies based on the authorization level of the client. Upon completion of the operation, the server sends back the result of the operation along with any output (**stdout** and **stderr**) generated in the process.

The server's built-in command language provides the building blocks for developing high-level administrative tasks. The ability to easily add new commands to the server's interpreter lets administrators create their own tasks tailored for their particular installation, and assign specific lists of users who are authorized to run them. Once a command is defined in a sysctl server, it is accessible to any authorized user from any workstation.

The Tcl Command Language

Sysctl uses the Tcl[2][3] embeddable command language as the foundation for its built-in interpreter. The Tcl library contains a parser for a simple command language and a collection of built-in utility commands. It also has a C interface that host programs can use to augment the built-in set of Tcl commands with application specific commands. The use of an embedded language, and Tcl in particular, provides several advantages:

- A command language interface is ideal for administrative tasks.
- The inherent extendibility of the Tcl interpreter allows the sysctl language to be extended without recompiling C code.
- The use of a command language lets clients send over *scripts*, rather than just discrete commands. This gives a high level of flexibility as new operations can be created dynamically by combining lower level commands into high-level tasks without having to pre-distribute or pre-register scripts.
- The use of an embedded language gives the sysctl server complete control over the set of commands available to a client. This control forms the basis of the server's multi-level authorization scheme.

Authorization

One of the traditional problems with UNIX administrative tools is that while they may have useful features, they usually don't have robust authorization mechanisms. At Project Agora, the goal is to develop tools and procedures for administering multi-campus, enterprise-wide environments. It is in these large environments where security and authorization become particularly relevant. The sysctl package's authorization scheme was designed specifically to solve the *all-or-nothing* authorization syndrome of programs like *rsh*(1) and others.

Kerberos and Access Control Lists

An authorization scheme needs a reliable *authentication* service in order to confirm the identity of a user. The sysctl package uses the Kerberos authentication service to provide trusted third-party authentication of users. The sysctl request sent from the client contains a Kerberos ticket that uniquely identifies the user that initiated the request. Given an authenticated identity, the server uses Access Control Lists (ACLs) to determine authorizations. Access Control Lists serve two purposes within the server:

- Identifying to the server its set of trusted users.
- Controlling access to sysctl commands.

A sysctl ACL is a plain text file. Each line of an ACL file is interpreted as either a Kerberos principal name or the name of another ACL. When an

ACL is searched for a principal name, files listed within the ACL are also checked. This hierarchical structure makes it easier to maintain customized sets of authorized users on certain machines while still maintaining a global default set of authorized users. For example, an ACL stored in AFS can contain a core set of system administrators and also the name of a local file that can be customized per machine.

Authorization Levels

Each sysctl server has one ACL that lists its trusted users. This ACL is used to determine a client's *base authorization level*. When a sysctl request is received, the server assigns it one of three authorization levels:

- **Unauthentic**. No ticket was sent in the request or the identity of the client was not verified via Kerberos.
- **Authentic**. The client has been authenticated via Kerberos but is not listed in the server's ACL of trusted users.
- **Trusted**. The client is authenticated and is listed in the server's ACL of trusted users.

The server internally manages three separate interpreters that map to the three base authorization levels. These interpreters are configured to provide access to those commands authorized for its level only. When a request is received and the base authorization level of the requester is determined, the commands contained in the request are passed to the appropriate interpreter for execution. Thus, the set of commands available to a user at a particular authorization level are restricted to the commands authorized for the associated interpreter.

Task Authorization

While the server's authorization levels prevent unauthorized users from directly executing certain sysctl commands, it is sometimes desirable to authorize users to run a *task* that may contain commands they are not ordinarily able to run. This task authorization is accomplished by defining external procedures.

When an external procedure is defined in the server it is assigned an authorization. This can be either one of the three base authorization levels or a separate ACL.

External procedures can contain commands that are not directly accessible to a user. For example, it is possible to create a procedure which any unauthenticated user can run that contains the Tcl **exec** command. When an unauthenticated user runs the procedure, the **exec** contained in it will run successfully even though that user is not authorized to run **exec** directly.

Authorization Variables

Prior to executing the client request, the server sets several read-only variables within the interpreters that provide information on the authenticated identity of the client. These variables are mirrored with environment variables giving external scripts access to this information.

The purpose of the authorization variables is to provide a mechanism for external commands and procedures to create their own authorization checking mechanism. This is useful in cases where the desired authorization strategy does not fit an ACL structure. For instance, the authorization for a password changing command might be that a user is only authorized to change his own password. The variables provide the elements needed to perform this type of check.

Programming the Sysctl Server

One of the distinguishing features of the sysctl package is the ability to program the server component. In a sense, sysctl can be thought of as a general-purpose, authenticated client/server system that an administrator can *program* to perform whatever functions are required to manage the environment. In many cases, using sysctl is preferable to RPC programming because with sysctl, the programming is achieved using external configuration files rather than C programming.

The /etc/sysctl.conf File

The **/etc/sysctl.conf** file is the sysctl server configuration file. It is used to specify additional commands and procedures to be made available through the server and to override default configuration values. The configuration file modifies the state of the server either by assigning values to configuration variables or by executing one or more configuration commands.

The configuration commands create read-only variables, register external commands and procedures, include other configuration files, and create *classes* of external commands. The syntax of these commands is shown below.

`include` *filename*
`create var` *variable value*
`create cmd` *name auth* **yes|no** *command*
`create proc` *name auth args body*
`create class` *label filename*

Assigning values to the variables **ACL**, **LOG**, and **KEY** define the server's ACL file, the log file, and the file containing the server's Kerberos key, respectively. Environment variables, such as the default **PATH**, are set by assigning values to the **env()** array.

The include Command

The **include** command specifies additional configuration files to be read by the server. This makes it easier to manage large numbers of external command definitions by allowing them to be split among a hierarchical set of files.

The create var Command

The **create var** command defines read-only variables in each of the three interpreters. Variables defined in the configuration file using the standard **set** command will only exist while the configuration files are being read.

The create cmd Command

The **create cmd** command makes external shell commands available as sysctl commands through the server. Four arguments are required for this command:

- **name**: The name by which the command is created inside the interpreters.
- **auth**: An authorization specifier for the command. This can be **NOAUTH**, **AUTH**, or **ACL** to assign it one of the built-in authorization levels, or it can be the path name of an ACL file.
- **yes|no**: Indicates whether or not command arguments from the client request are to be passed to the shell command. The parameters passed to the shell command are not interpreted directly by the server but special shell characters (e.g., *;|()['']$<>\n&+*, etc.) are not allowed in any of the parameters. If any of these characters are found the command is rejected.
- **command**: The shell command(s) to execute. This can contain any valid shell (**/bin/sh**) syntax, including multi-line commands.

You can think of external commands as embedded shell scripts in the server. The following example defines a sysctl command to clean out the **/tmp** directory. Any authenticated user is allowed to run this command and no arguments will be passed to the shell invocation.

```
create cmd clean_tmp AUTH no {
  find /tmp -type f \
      -mtime +2 -atime +2 \
        | xargs rm -f
}
```

The create proc Command

The **create proc** command defines external sysctl procedures. This command requires four arguments:

- **name:** The name by which the command is created inside the interpreters.

```
set dir /vol/sysctl/src/sample/files

# These declarations override the default locations
set ACL /etc/sysctl.acl
set LOG /usr/adm/sysctl.log
set KEY /etc/srvtab

# Set an environment variable
set env(PATH) /bin:/usr/bin:/usr/ucb:/etc:/usr/etc

# These variables are set in the service interpreters and marked readonly
create var ARCH [exec /bin/arch]
create var VERSION [exec /etc/agora/version]

# Include command class files
create class sample $dir/sample.cmds

# Create a command
create cmd date NOAUTH no {
     /bin/date
}
# Create a procedure
create proc sysinfo AUTH {} {
     global SCLHOST ARCH VERSION
     echo $SCLHOST $ARCH $VERSION
}

# Check for local configurations
if [file exists /etc/sysctl.conf.local] {
     include /etc/sysctl.conf.local
}
```

Figure 1: Sample **/etc/sysctl.conf** File

- **auth:** An authorization specifier for the command. This can be **NOAUTH**, **AUTH**, or **ACL** to assign it one of the built-in authorization levels, or it can be the path name of an ACL file.
- **args:** A list of names of arguments to the procedure.
- **body:** A sysctl (Tcl) expression that forms the body of the new procedure.

Executing a sysctl procedure is usually more efficient than executing an external shell command since the server does not have to *exec*(3) a shell to execute the command. This example defines a procedure to report filesystem usage. Only users listed in the ACL **/etc/fs.acl** are authorized to run this command. The name of the filesystem to check is passed as an argument. The variable **SCLHOST** is one of the authorization variables set by the server.

```
create proc fsinfo /etc/fs.acl {fs} {
    global SCLHOST
    statfs $fs buf
    echo "Filesystem $fs on $SCLHOST:"
    echo "Size: $buf(size) KB"
    echo "Used: $buf(used) KB"
    echo "Free: $buf(free) KB"
}
```

The create class Command

The **create class** command defines classes of commands within the server. Command classes provide a way to organize commands into logical groups for clarity. The **label** parameter specifies a *tag* that is prepended to all commands defined in the command file. If a class named **test** is created and the class file defines a procedure named **help**, the procedure is created in the interpreters as **test:help**.

Sysctl Client Library

The sysctl client library, **libsysctl.a**, provides a C language interface for communicating with **sysctld** servers. A full description of the programming interface is available in Appendix A. The main routine in the library is the **sysctl()** function:

```
#include <sysctl.h>
sc_result *sysctl(char *host,
                  char *op,
                  sc_control *cntl)
```

The **host** parameter is a pointer to a string that contains the name of the host to contact. The **op** parameter is a pointer to a string that contains the operation to run on the remote server. The **op** string can contain any sysctl expression and can be anything from a single command with no arguments to an entire script. The **sysctl()** routine does not interpret the **op** parameter. The **cntl** parameter is a pointer to an **sc_control** structure. This structure controls the communication mode between the client and server and is used to determine whether information is encrypted before it is transmitted, connection and operation timeouts, the port to use to connect to the remote server, etc.

Before sending the operation to the remote host, **sysctl()** attempts to obtain a Kerberos ticket for the service **rcmd.***hostname*, where *hostname* is the fully qualified hostname[1] of **host**. If a ticket cannot be obtained, the operation is sent to the server without any authentication information.

The result of the operation is returned via an **sc_result** structure. This structure contains the exit status of the remote operation as well as any output generated. See Appendix A for more details.

Sysctl Client Command

The sysctl client command provides a command-line interface to the sysctl package. All of the features of the **sysctl()** library routine are accessible via command line arguments. It is ideally suited for integrating sysctl calls into shell or perl scripts. A complete listing of the command options is given in Appendix C.

Integrating Existing Applications

One of the strengths of the sysctl package is the ease with which it integrates with existing tools. The following example illustrates the steps necessary to convert a typical administrative perl script designed for stand alone operation to work with sysctl.

The example script is called **chmbox** and is used to change a user's mail alias in the file **/usr/lib/aliases**. It takes a username and the new mail destination as arguments. Using sysctl, this same script can be used to let users update their mail alias remotely in a global alias file. The command only allows users to change their own mail alias. The host that houses the global alias file is called **admin**. Listed below are the steps required to set this up:

1. Modify **/etc/sysctl.conf** on **admin** to register the **chmbox** command.
2. Add authorization checks to the existing **chmbox** script.
3. Restart **sysctld** on **admin** to activate the command.

First, register the command. For this example, the entry is placed in **/etc/sysctl.conf**. If there were a suite of mail service commands, it would probably make sense to create a separate command class file.

```
sysctl -h admin 'confadd {
create cmd chmbox AUTH yes /etc/chmbox
}'
```

[1]Defined as the value returned by the *gethostbyname*(3) system call.

The **confadd** command is used to modify the contents of a server's configuration files. The **create cmd** command defines an external command named **chmbox**, and associates the script **/etc/chmbox** with it. The **AUTH** value in the authorization field ensures that only authenticated users can run this command. Additional authorization checks are performed within the modified **chmbox.** The **yes** value in the parameter field indicates that **/etc/chmbox** will accept arguments.

Next, add an authorization check to the existing **/etc/chmbox** script. This check is inserted after the arguments have been processed but before any files are modified. The check fails if the user name given in the command-line does not match the authenticated name retrieved from the sysctl environment variables:

```
# Assume the name of the alias to
# change is in $user
$ruser = $ENV{'SCUSER'};
$rrealm = $ENV{'SCREALM'};
$lrealm = $ENV{'SCLREALM'};

if (($user ne $ruser) ||
   ($rrealm ne $lrealm)) {
      die "Permission denied.\n";
}
```

Finally, restart the server on **admin** by issuing the command:

```
sysctl -h admin svcrestart
```

Users can now change their mailbox by issuing the command:

sysctl -h admin chmbox *user mbox*

Better yet, create an executable script that contains the following lines:

```
#!/usr/agora/bin/sysctl -r
#host#admin

if {$argc != 2} {
  error {Usage: chmbox <user> <mbox>}
}
set user $argv(1)
set mbox $argv(2)

if {[regexp {(.+)@(.+)} $mbox] != 1} {
  error {Alias must be user@host}
}
chmbox $user $mbox
```

The **-r** (replay) option to **sysctl** is used to send an entire script over to a server. It requires a file name argument. In this example, we take advantage of the **#!** specifer which executes **sysctl** with the replay file name argument (the script name) appended. The **#host#** syntax in line 2 of the script allows a default server to be set. Ordinarily, if no host is specified on the command line via the **-h** or **-c** flags, **sysctl** sends the operation to the local host. But if **-r** is used, **sysctl** looks at the first line of the replay file (unless it starts with **#!** in which case it looks at the second line) and checks for the keyword **#host#**. If the keyword appears, **sysctl** interprets the word that follows it as the name of a host and sends the script to that host. This in effect produces a *transportable* script! There is also support for a **#hesiod#** keyword which causes **sysctl** to query the Hesiod name server to find the name of a server to use and a **#cluster#** keyword which causes the script to be run on all machines in the specified cluster.

Another feature demonstrated in the script is the use of the **argc** and **argv()** variables. When run with -r, **sysctl** treats all extra command line arguments (i.e., those not associated with a flag) as arguments to pass to the replay file and sets the **argc** and **argv()** variables in much the same way as the *exec*(3) family of C routines.

Sysctl Applications

The method of writing a sysctl application generally consists of the following steps:

1. Examine what you are trying to do and divide it along client/server lines.
2. Define the client/server interface which consists of sysctl command names and their arguments, return values, and authorizations.
3. Code the client front end. This can be a fancy GUI, a perl or shell script, or even a simple sysctl script. The front end normally collects some information before making the appropriate sysctl call(s).
4. Code the server sysctl commands. For relatively small items, these are coded as external sysctl procedures. For more complex things, perl or shell scripts are written and inserted into the server as external commands or with "glue" procedures.

The rest of this section describes a **Home Directory Mover** sysctl application used at Agora. Agora currently supports user home directories located either in central AFS storage or on NFS-mountable volumes on a user's private workstation. The application, named **mvhomedir,** provides a seamless mechanism for Agora system administrators to move a home directory from its current location to a new location. The requirements for the design were:

- The **mvhomedir** command must be executable only by a subset of Agora system administrators.
- The administrators are not authorized to directly create, delete, or modify NFS or AFS volumes, or any data file that gets modified during the moving of a home directory, such as automounter maps.
- The administrators are authorized to transfer data between machines only for the purposes of moving a user's home directory.

The basic design of the application is shown in the accompanying diagram. The application defines two sysctl command classes: **mvh** and **mvh_server.** The **mvh** class contains one command, **mvh:mvhomedir**. This command accepts arguments that give the user name, the current location of the home directory, and the new location. Either location can be in AFS or NFS. The **mvh_server** class contains a number of commands that actually move the home directory.

A front end program queries the administrator for information and then runs the sysctl command **mvh:mvhomedir** to a secure server. The sysctl server on the secure system only executes the request if the issuer is on the **mvh** ACL. If the ACL check passes, the secure server runs two sysctl calls as itself[2] on behalf of the administrator. The first call (**mvh_server:nfstarfrom**) extracts the data in the user's current home directory, and the second (**mvh_server:nfschome** and **mvh_server:nfstarto**) creates the new home directory and unpacks the files. Both of these backend sysctl calls are run in *socket* mode to handle the bulk transfer of data. The source and destination hosts both check to make sure the principal executing the tar commands (in this case the secure server) is authorized by checking the **mvh_server** ACL. The **mvh_server** command class also has a suite of commands that perform the same functions for AFS: **mvh_server:afschome, mvh_server:afstarto**, and **mvh_server:afstarfrom.**

Note that the **mvh** and **mvh_server** command classes have separate ACLs associated with them. The only principal listed in the **mvh_server** ACL is **rcmd.***host*, where *host* is the hostname of the secure server. This means that although the administrators are authorized to run the high-level **mvhomedir** command, they do not have the ability to directly run the tar commands or the commands which create the user's home directory volume.

The entire application consists of two sysctl command files that contain the application's sysctl command definitions (about 80 lines of code), a front end script that collects and sanity-checks information from the administrators and then issues the sysctl call (about 500 lines of perl code), and a set of back-end perl scripts to handle complex tasks such as creating AFS volumes and mount points, modifying administrative files, etc. (total of about 850 lines of perl code).

Future Directions

The sysctl client and server code have been compiled and run under AIX, HPUX, SunOS, and IRIX. A port of the client code has been made to a mainframe VM system. Porting of the client and server code to lower-end platforms such as OS/2 and DOS/Windows would increase the viability of sysctl for use in administering a truly heterogeneous environment.

There are two items which present some opportunity for improvement to sysctl. The first would be to remove the ACL processing code from the server and install it into a separate C library. This would allow the server and other applications to share the same basic ACL structure. It would also be advantageous to allow ACLs to be retrieved from authorization servers and include a more robust set of objects, such as IP addresses, etc.

Second, it may be worthwhile to port the client and server to use the secure RPC available in the OSF/DCE product. The ACL service provided as part of DCE may also be relevant to the first item above.

[2]It uses the **-l** flag to the sysctl client command.

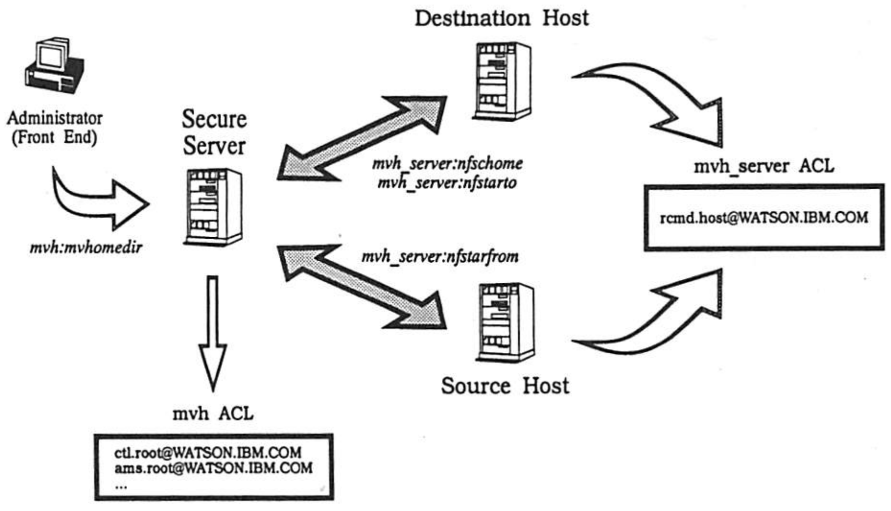

Figure 2: Home Directory Mover Design

Acknowledgements

The authors would like to acknowledge the contribution of Mike Dugan, who suggested rolling Tcl into the server. Also thanks to Mark Linehan, Aaron Shtromas, Jerry Larivee, and Lucien Van Elsen for their comments and suggestions on the design. The **Home Directory Mover** application was designed and implemented by Geetha Vijayan.

Availability

For information about the availability of **sysctl,** send mail to **agora-info@watson.ibm.com**.

Author Information

Salvatore DeSimone is a member of the System Architecture Group within Project Agora. Christine Lombardi is a member of the Agora Software Engineering Group. Their e-mail addresses are vatore@watson.ibm.com and ctl@watson.ibm.com. They can be reached via U.S. Mail at the IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY, 10532.

References

[1] Jennifer G. Steiner, Clifford Neuman, Jeffrey I. Schiller, *Kerberos: An Authentication Service for Open Network Systems*, Proc. USENIX Winter Conference, January 1988.

[2] John K. Ousterhout, *Tcl: An Embeddable Command Language*, Proc. USENIX Winter Conference, January 1990.

[3] Karl Lehenbauer, Mark Diekhans, *Extended Tcl – Extended Command Set for Tcl 6.2*, unpublished manual page, January 1992.

[4] Stephen P. Dyer, *The Hesiod Name Server*, Proc. USENIX Winter Conference, 1988.

[5] Stephen Shafer and Mary Thompson, *The SUP Software Upgrade Protocol*, Carnegie Mellon University, School of Computer Science, 1988.

[6] Larry Wall, Randal L. Schwartz, *Programming perl*, O'Reilly and Associates, Sebastopol, CA, 1990.

Appendix A – Programming Interface

```
#include <sysctl.h>
sc_result *sysctl(char *host,
                  char *op,
                  sc_control *cntl)

int sysctl_errno;
char *sysctl_msg(int err)

int sc_ReadMsg(sc_sock *sd)
int sc_WriteMsg(sc_sock *sd)
int sc_CloseSocket(sc_sock *sd)

typedef struct {
     int flags;
     int port;
     int conn_timeout;
     int timeout;
} sc_control;

typedef struct {
     int status;
     char *result;
     sc_sock *sd;
} sc_result;

typedef struct {
     int sockfd;
     u_short msgtype;
     u_long msglen;
     char *msg;
     u_short status;
} sc_sock;
```

The sysctl() Routine

The **sysctl()** routine is a C library interface for communicating with **sysctld** servers. The **host** parameter is a pointer to a string that contains the name of the host to contact. The **op** parameter is a pointer to a string that contains the operation to run on the remote server. The **op** string can contain any sysctl expression and can be anything from a single command with no arguments to an entire script. The **sysctl()** routine does not interpret the **op** parameter. The **cntl** parameter is a pointer to an **sc_control** structure.

Before sending the operation to the remote host, **sysctl()** attempts to obtain a Kerberos ticket for the service **rcmd.***hostname*, where *hostname* is the fully qualified hostname of **host**. If a ticket cannot be obtained, the operation is sent to the server without any authentication information.

The result of the operation is returned via an **sc_result** structure. This structure contains the exit status of the remote operation as well as any output generated.

The sc_control Structure

The **sc_control** structure is used by the caller to control the communication mode between itself and the remote server. The **flags** field contains a bit-OR'ed set of flags, as defined in **sysctl.h**:

- **SC_NOWAIT**. The caller is not interested in the results of the remote operation. The server sends an immediate acknowledgement back to the caller and discards the result of the operation. Note that the acknowledgement is sent back after the authentication checks so any errors related to decoding Kerberos tickets are relayed to the client.
- **SC_NOAUTH**. No authentication information should be sent to the remote server.
- **SC_SOCKET**. The server is to return the result to the client through a TCP socket, rather than in the RPC return structure. This is useful in cases where large amounts of data are to be returned[3], when binary data is returned, or when the client wishes to send data to the remotely executing process.
- **SC_INTERACTIVE**. This is similar to **SC_SOCKET** mode in that a TCP communication stream is established between client and server. However, in interactive mode the client is connected directly to the remote interpreter rather than to an executing command. The **op** parameter is ignored in this case.
- **SC_PRIVATE**. Encrypt all communications between the client and server.
- **SC_NULL**. This flag causes **sysctl()** to make an RPC call to the **NULL** procedure on the remote server. The **op** parameter is ignored in this case. This provides a quick mechanism for determining if a **sysctld** server is up and running.

The **conn_timeout** field determines the length of time sysctl() waits for the initial connection to the server to be established. The **timeout** field is used to specify the maximum total timeout for the operation. If this maximum timeout is reached, **sysctl()** closes the connection with the remote server (which causes a **SIGHUP** signal to be sent to all remote processes) and returns an error. These timeouts only affect the RPC communication with the server and have no impact on the socket transfers if the caller specified **SC_SOCKET**. In all cases, the value of the timeout variables are interpreted as seconds. If they are set to zero, default values are selected.

The **port** parameter specifies the TCP port to use to connect to the remote server. If the port number passed is 0, **sysctl()** uses the *getservbyname*(3) system call to get the port number.

If **sysctl()** receives a **NULL** value for the **sc_control** parameter, it assigns default values for all fields. These default values are: **flags** = 0, **conn_timeout** = 10, and **timeout** = 1800.

[3]The maximum size of the result that can be sent back via the RPC structure is currently set at 1 MB.

The sc_result Structure

The **sc_result** structure is used to return information about the remote operation to the caller. The **status** field contains the exit status of the remote operation. The **result** field is a pointer to a string that contains the output of the remote operation, or a **NULL** value if the **SC_SOCKET** or **SC_INTERACTIVE** flags were specified. In the case of **SC_SOCKET** and **SC_INTERACTIVE**, the **sd** field contains a pointer to an **sc_sock** structure which is used to transfer data to and from the server.

Error Handling and Return Codes

If an error occurs during the transmission of a request to the server, **sysctl()** returns **NULL** and sets the global variable **sysctl_errno** to indicate the error. The **sysctl_msg()** library routine is used to return a pointer to a descriptive error message.

Otherwise, **sysctl()** returns a pointer to an **sc_result** structure. The exit status of the remote operation contained in the **status** field is determined as follows:

- If the remote server was unable to execute any portion of the operation, perhaps due to a syntax error or insufficient authorization, a value of **SCERR_TCLERROR** is returned in **status**. The **result** field contains any output produced up to the point the error occurred with the last line containing an error message describing the problem.
- If the remote operation was terminated by an **exit** statement, **status** contains the value of the parameter passed to **exit**.
- The shell exit status of the last external command executed, otherwise a value of 0.

The sc_sock Structure

The **sc_sock** structure is used in **SC_SOCKET** and **SC_INTERACTIVE** modes to transfer data between client and server. The **sc_WriteMsg()** and **sc_ReadMsg()** routines implement a simple message passing scheme that allows data to be sent to and received from the remote server. The use of message passing gives the client the ability to send and receive binary data, demultiplex **stdout** and **stderr** output from the remote process, and retrieve the exit status of the remote operation.

The **sc_ReadMsg()** routine is used to read messages from the server. It is passed the **sc_sock** pointer returned in the original **sysctl()** call. If an error occurs while reading from the socket, a value of -1 is returned and **sysctl_errno** is set to indicate the error. Otherwise, the **msgtype** field indicates the type of message received. The valid message types are:

- **SC_MSGSTD.** The **msg** field points to a static buffer of **msglen** bytes of **stdout** from the server.
- **SC_MSGERR.** The **msg** field points to a static buffer of **msglen** bytes of **stderr** from the server.
- **SC_MSGEOF.** The remote operation has terminated. The **status** field contains the exit status of the remote operation. The status field is interpreted in the same way as the exit status returned in the RPC structure.

The **sc_WriteMsg()** routine is used to send data to the remote server. The client must fill in the **msgtype**, **msg**, and **msglen** fields of the **sc_sock** structure. **msg** should point to a buffer containing **msglen** bytes of data to send to the server. If the message type is **SC_MSGSTD** or **SC_MSGERR**, the data sent appears as **stdin** to the remote process. If the message type is **SC_MSGEOF**, the server closes **stdin** of the remote process. The **msg** and **msglen** fields are ignored when sending an **SC_MSGEOF** message. If an error occurs while writing to the socket, **sc_WriteMsg()** returns -1 and sets **sysctl_errno** to indicate the error.

Note that if the **SC_PRIVATE** flag was set in the original **sysctl()** call, all messages to and from the server are encrypted. The encrypting and decrypting of the message data is handled within the **sc_ReadMsg()** and **sc_WriteMsg()** routines.

Appendix B – Code Samples

```
$ cat whoami.c
/*
 * This code sample takes the first argument to be the host to contact.
 * It uses the command "whoami" to echo the authenticated identity of the
 * user. The communication between the client and server will be encrypted.
 */
#include <sysctl.h>

#define CMD "echo Server $SCLHOST says I am [whoami]"

main(int argc, char *argv[])
{
     sc_control cntl;
     sc_result *r;

     bzero(&cntl, sizeof(sc_control));
     cntl.flags |= SC_PRIVATE;

     r = sysctl(argv[1], CMD, &cntl);
     if (r == NULL)
          printf("sysctl error: %s\n", sysctl_msg(sysctl_errno));
     else
          printf("%s", r->result);
}
$ cc -o whoami whoami.c -lsysctl -lkrb -ldes
$ whoami harlem
Server harlem.watson.ibm.com says I am vatore.@WATSON.IBM.COM
$
```

```
$ cat fscheck
#!/usr/agora/bin/sysctl -Lr
#
# Check filesystem usage - high-water pct mark is passed as a parameter
#
foreach fs [listfs] {
        statfs $fs sbuf
        set pct [expr $sbuf(used).0/$sbuf(size).0*100]
        if {$pct > $argv(1)} {
                set pct [format "%.2f" $pct]
                echo "$fs ==> ${pct}% Used, $sbuf(free) KB Free"
        }
}
$ cat /tmp/hostlist
harlem
badger
$ time fscheck -c /tmp/hostlist 80
harlem::/ ==> 84.29% Used, 1538 KB Free
badger::/usr ==> 98.35% Used, 3656 KB Free

real    0m0.29s
user    0m0.05s
sys     0m0.06s
$
```

Appendix C – Sysctl Command Usage Information

Usage: **sysctl** [*options*] [*command* ...]

Options:

-c *cluster*
: Run the command on the specified cluster. If the *cluster* argument contains a "/", it is assumed to be a file that contains the names of hosts in the cluster, one per line. Otherwise, the list of hosts in the cluster is retrieved by a call to Hesiod. More than one cluster can be specified with multiple -c options.

-f *num*
: Controls maximum fan-out. By default, a maximum of four concurrent connections is used. This allows greater (or lesser) parallelism.

-h *host*
: The server to execute the command on. More than one host can be specified using multiple -h options. If no -h (or -c) option is specified the server is assumed to be the local host.

-i *instance*
: Authenticate as this instance. The user is prompted for a password.

-L
: Provides an alternate way of delimiting output from multiple servers. Prepends each line of output from a host with the hostname, such as **foo.watson.ibm.com::....**

-l
: Authenticate as the local server. The program uses the sysctl server key stored in the local Kerberos keyfile (**/etc/srvtab**) to obtain a ticket. This authenticates the user to the sysctl server as **rcmd.***hostname*, where *hostname* is the local hostname. This is used to allow the local **root** user to gain access to sysctl commands reserved for authenticated users if the command is invoked by programs such as **cron**. The user must be running as **root**.

-n
: Send no authentication information.

-P *port*
: Specify the port number to use to connect to the remote server.

-p
: Use private (encrypted) communication.

-q
: Quick mode - Don't wait for the result from the server.

-r *file*
: Replay (drop) this file on the server(s).

-s
: The server should send the results back via a TCP socket. The output from the server is de-multiplexed into **stdout** and **stderr**.

-t *sec*
: Specify the connection timeout.

-T *sec*
: Specify the (RPC) remote operation timeout.

-u *user*
: Authenticate as this user. The user is prompted for a password.

-x
: Send a NULL RPC to the server(s) (like a ping).

command ...
: The command(s) to pass to the server.

If no command is given **sysctl** runs in interactive mode. The results of the server execution are printed to **stdout**. In the case of multiple servers, the output is delimited on a server-by-server basis for easy parsing.

Appendix D – Sysctld Command Usage Information

Usage: **sysctld** [*options*]

Options:

-A
: Enable default authorization of ***.admin** principals.

-a *file*
: Specify the name of the server ACL file (default: **/etc/sysctl.acl**).

-d
: Run in debug mode. This causes more information to be printed to the log file and is useful for debugging problems with the server.

-k *file*
: Specify the name of the file that contains the Kerberos server key (default: **/etc/srvtab**).

-l *file*
: Specify the name of the log file (default: **/usr/adm/sysctl.log**). If the file specified is "syslog", the server logs all messages to **syslogd**.

-n
: Run the server with authorization turned off. This should only be used for testing.

-P *port*
: Specify the port number at which to register the service.

Any options given on the command line override the contents of the server configuration file.

Automated System Monitoring and Notification With Swatch

Stephen E. Hansen & E. Todd Atkins – Stanford University

ABSTRACT

This paper describes an approach to monitoring events on a large number of servers and workstations. While modern UNIX systems are capable of logging a variety of information concerning the health and status of their hardware and operating system software, they are generally not configured to do so. Even when this information is logged, it is often hidden in places that are either not monitored regularly or are susceptible to deletion or modification by a successful intruder. Also, a system administrator must often monitor several, perhaps dozens, of systems. To address these problems, our approach begins with the modification of certain system programs to enhance their logging capabilities. In addition, our approach calls for the logging facilities on each of these systems to be configured in such a way as to send a copy of the critical system and security related information to a dependable, secure, central logging host system. As one might expect, this central log can see a megabyte or more of data in a single day. To keep a system administrator from being overwhelmed by a large quantity of data we have developed an easily configurable log file filter/monitor, called *swatch*. Swatch monitors log files and acts to filter out unwanted data and take one or more user specified actions (ring bell, send mail, execute a script, etc.) based upon patterns in the log.

The Problem

It is an unfortunate fact that most UNIX systems, as delivered, do little to ease the job of the system administrator when it comes to keeping tabs on the health of those systems. Often, the first inkling of a problem occurs when keystrokes stop being echoed or the phone rings.

What every good system administrator tries to do is keep an eye on the health of each of the systems in his or her care. The health of a system should be reflected in the log messages generated by the kernel and the various daemons and utilities. In addition, these messages should also include information relevant to system security. However, with most systems we have seen, the system's log information is not generally made available to the system administrator in a way that is either secure or convenient, rather it is often hidden in places that are either not monitored regularly or are susceptible to corruption or destruction by system failure or a successful intruder. The assumption seems to be that system log files are only to be consulted after the fact, to help with postmortem rather than prevention. What this means is that the UNIX *syslog*(3) facility, regardless of the original intent, is generally used as more of a debugging aid than as a tool for system management.

Improved Security Logging

For purposes of monitoring systems security, standard UNIX logging features often prove to be inadequate and/or inconvenient. To address this problem, our approach begins with the modification of certain system utilities to enhance the reporting done, particularly with regard to possible security related activities. Table 1 lists some of the utilities modified and the changes made to their logging capabilities.

Program	Logging Enhancements
fingerd	Reports the originating host and the *finger* target(s) to syslog.
ftpd	Reports originating host to syslog. Reports file transfers to a local log file along with the local user name and, if the user is "anonymous", the password.
ruserok	Used by *rshd* and *login* when called by *rlogind.* Disallows and reports to syslog any attempts to use a */etc/hosts.equiv* or *~/.rhosts* file that contains a '+'.
rshd	Reports the access status, local user, remote user and host, and the command issued to a local log file.
login:	Reduced number of tries to three. Reports to syslog on 'Incomplete Login Attempt', 'Repeated Login Attempt', and 'Root Login Refused'. Includes the account names attempted and the originating host.

Table 1: List of logging enhancements made to several system programs.

At our site we were fortunate enough to have access to the vendor's source code for all our utilities. While this is not possible for everyone, each of the utilities listed in Table 1 are available from various network archive sites. In a few cases it might be preferable to use the public version instead to improve portability. Another source of security related information is from the tcp wrapper code written by Weitse Venema[1]. Besides providing access control for those network services run out of *inetd*, it generates information via syslog about the connections it mediates.

One important utility not listed in Table 1 is *sendmail*(8). Even without modification sendmail can be configured to generate a plethora of status information. Unfortunately, *sendmail* isn't very discriminating in what it reports, assigning every status message the same priority.

Centralized Logging with syslog

When we have added to the logging capabilities of the various utilities, we have, for the most part, made use of the syslog library functions. Besides providing a consistent and relatively standard logging interface, syslog directs logging messages to different files or hosts based upon the source of the message and its level of importance.

The way our facility is set up, each server system keeps its own copy of most of the syslog messages in the file */var/log/syslog*. These syslog files are rotated on a daily basis, compressed, and kept online for about a week. Log messages that might reflect a system's health or potential security problems are also forwarded to a central log host, the LOGMASTER. In practice this means that almost everything except sendmail status messages are sent to the LOGMASTER. Leaving the sendmail status messages on the servers cuts down on the network traffic due to syslog without significantly affecting our ability to monitor. On our systems the sendmail messages can account for as much as 90% of a host's log messages, although 50% is more common. Appendix A shows the syslog configuration file (*/etc/syslog.conf*) for a host being monitored. The last three lines in the file are responsible for sending data to the LOGMASTER.

Copying the syslog information to a central site is done for several reasons. First, it provides redundancy and security. If the log files on the originating host are destroyed or modified, either accidently or by malicious intent, those on the more secure central site will be left intact. Second, it simplifies the monitoring of all the log information. By collecting information from a number of systems in a single time ordered file, problems may be found that would be missed if viewed in isolation, such as network or security related problems. For example, a single failed log in attempt on one system might be attributed to a typing error. The same failed log in attempt occurring on several systems in sequence could indicate an intruder trying to break in. Collecting information from several different system utilities as well as from more than one system can provide information indicating a pattern of attack. Several *fingers* followed by a failed *login* or *rsh* command is a common pattern revealed by this type of monitoring.

Winnowing the Chaff: An Introduction to Swatch

Our facility manages about a dozen file and CPU servers which have over 50 client machines. The server systems receive an enormous amount of log information through the syslog daemon. Even after filtering out the sendmail information messages the LOGMASTER sees about a megabyte of syslog messages per day. As one can imagine, sorting through that much information on a daily basis can be very time consuming. We also found that some important log entries tend to get lost among all of the less important entries when examining the log files.

One solution to this problem would be to search for certain types of information, which can be done by using the *egrep*(1) program with some complex command line arguments. Even with this solution one still has the problem of having to constantly monitor the output so that the urgent information is seen when it comes in. Some of this information needs to be acted on soon after it is received. For example, if the system on a file server machine locks up then somebody needs to be alerted so the machine can be brought back up as quickly as possible. For us the most desirable solution was to have a more complex program sift through the log and do a few simple tasks when certain types of information were found. We decided to call this program *swatch*, which stands for *S*imple *WATCH*er.

Swatch Design Goals

There were four goals that were set when designing swatch.

1. Configure the program in such a way that it would only take a few minutes to teach any systems administrator how to use it.
2. Have a simple set of actions that could be performed after receiving certain types of information.
3. Allow swatch's users to define their own actions if they like, and allow them to use parts of the input as arguments to the action.
4. Once swatch is running it should be reconfigurable on demand or after a specified interval without having to stop and restart the program manually.

Using Swatch

Swatch may be run three different ways: make a single pass through a file; look at messages that are being appended to a file as that file is being updated; or examine the standard output of a program. A complete description of swatch's command line options can be found in Appendix B.

Swatch's most powerful function is in examining information as it is being appended to a log file. We use swatch to look at messages as they are being added to the syslog file, alerting us immediately to serious system problems as they occur. Using a *tail*(1) of */var/log/syslog* as input is the default action for swatch but another file can be "tailed" by using the -t command line option as in

```
swatch -t /var/log/authlog
```

Receiving timely notification of certain types of probes or attacks often enables us to find out which users are logged on to the originating system. Finding out such information can help identify hackers or compromised accounts.

By using the -f option, swatch can be made to read in and process a file from beginning to end. This single pass feature can be used to examine old syslog or other text files.

```
swatch -f /var/log/syslog.0
```

This option can be used to catch up on the contents of log files after being away from the computer for a while (like after vacationing in Hawaii for a week). This feature is often used to filter through several megabytes of old syslog files to look for evidence of suspected system and network related problems as well as system probes and break-in attempts.

Having swatch examine the output from a program is also useful. For example, one might want to sort through process accounting or other audit information that is not kept in a plain text file and requires special processing to read.

```
swatch -c swatchrc.acct -p lastcomm
```

Implementation

Swatch relies heavily on expression matching. For this reason the Perl[2] language was used because of its Awk and C like characteristics, as well as its increasing familiarity among systems administrators.

Swatch has three basic parts: a configuration file, a library of actions, and a controlling program.

Configuration File

Each non-comment line in a swatch configuration file consists of four tab separated fields: a pattern expression, a set of actions to be done if the expression is matched, an optional time interval, and the location of a time stamp, if any. As shown in Figure 1, a line's pattern field consists of one or more comma separated expressions while the action field may contain one or more comma separated actions.

The patterns must be regular expressions which Perl will accept, which are very similar to those used by the UNIX egrep program. Each string to be matched is compared, in order, with the expressions in the configuration file and if a match is found the corresponding actions are taken. A copy of the UNIX manual page for swatch's configuration file is listed in Appendix C.

The time interval can be used to help eliminate redundant messages. For example, on our systems `"file system full"` messages tend to come at the rate of several dozen per minute. We specify an interval of five minutes which will usually eliminate hundreds of redundant notifications.

The time stamp location information is optional and can only be used when a time interval is specified. Swatch uses it to strip away the time stamp in order to compare it to other messages which are stored in its internal history list.

Lines beginning with the '#' character are treated as comment lines and are ignored.

Actions

Swatch understands the following actions: echo, bell, ignore, write, mail, pipe, and exec.

- The **echo** action causes the line to be echoed to swatch's controlling terminal. An optional mode argument causes the text to be shown in normal, bold, underscore, blinking, or inverse mode. Normal mode is the default.
- The **bell** action sends a bell signal (^G) to the controlling terminal. An optional argument specifies the number of bell signals to send, with one being the default.
- The **ignore** action causes swatch to ignore the current line of input and proceed to the next one. The ignore action is mainly useful early on in the configuration file to filter out specific unimportant information that would otherwise match a more general expression found later in the configuration file.
- The **write** and **mail** actions can be used to send a copy of the line to a user list via the write and mail commands.
- The **pipe** and **exec** actions were added to provide some flexibility. The pipe action allows the user to use matched lines as input to a particular command on the system. The exec action allows the user to run a command on the system with the option of using selected fields from the matched line as arguments for the command. A $N will be replaced by the

```
/pattern/[,/pattern/,...]    action[,action,...]    [ [[HH:]MM:]SS    [start:length] ]
```

Figure 1: Format of pattern action line for a Swatch configuration file.

Nth field in the matched line. A $0 or a $* will be replaced by the entire line.

See Appendix C for more details on the actions and their arguments.

Controlling Program

The controlling program is swatch, but the real work is done by a watcher process. Swatch's first task is to translate the configuration file into a Perl script. After creating the watcher script, swatch forks and executes it as the watcher process. The watcher script also contains two signal handlers. Upon receiving an alarm (SIGALRM) or hang-up (SIGHUP) signal swatch will terminate the watcher process, re-read the configuration file, and start a new watcher process. If a quit (SIGQUIT) terminate (SIGTERM) or interrupt (SIGINT) signal is received, swatch will attempt to cleanup after itself then exit.

Examples

We have previously described several ways in which swatch can be used. In this section we will illustrate the two most common ways in which swatch is used at out facility. First, we have a swatch job running continuously looking for failed login attempts and system crashes and reboots. The swatch configuration file we use for this purpose is shown in Figure 2.

Second it's common for each system administrator to have a customized swatch configuration file in his or her home directory, ~/.swatchrc, that contains pattern/action pairs that are personally interesting, or that pertain to his or her system responsibilities. A swatch job using this configuration file is generally run in a window while the administrator is logged in. The personal swatch configuration file of one of the authors is shown in Figure 3, while Figure 4 shows six hours of output generated by this configuration.

Example 1: Continuous Monitoring for High Priority Events

This swatch configuration (Figure 2) runs in the background and continuously looks for high priority events, such as "file system full" and "panic" messages.

```
#
# Swatch configuration file for constant system monitoring in the background
#
# Test the pager every once in a while
/test pager/                  exec="/etc/call_pager 5551234 123"
#
# Bad login attempts
/INVALID|REPEATED|INCOMPLETE/ exec="/etc/backfinger $0"
#
# EECF
/EE-CF.*(panic|halt)/         mail=action,exec="/etc/call_pager 5551212 0911"
05:00   0:16
/EE-CF.*reboot/               mail=action,exec="/etc/call_pager 5551212 0411"
05:00   0:16
/EE-CF.*SunOS Release/        mail=action,exec="/etc/call_pager 5551212 0411"
05:00   0:16
/EE-CF.*file system full/     mail=action,exec="/etc/call_pager 5551212 0611"
05:00   0:16
#
# Sierra
/Sierra.*WizMON/              mail=action,exec="/etc/call_pager 5551234 1666"
05:00   0:16
/Sierra.*(panic|halt)/        mail=action,exec="/etc/call_pager 5551234 1911"
05:00   0:16
/Sierra.*reboot/              mail=action,exec="/etc/call_pager 5551234 1411"
05:00   0:16
/Sierra.*SunOS Release/       mail=action,exec="/etc/call_pager 5551234 1411"
05:00   0:16
/Sierra.*file system full/    mail=action,exec="/etc/call_pager 5551234 1611"
05:00   0:16
```

Figure 2: Swatch configuration file for continuous monitoring

The first pattern/action line is used to test our pager number periodically to ensure that swatch, our dial out line, and our pager are all working. We run the *logger*(1) program periodically via *cron*(1) to send a message which contains the string "test pager." This causes swatch to attempt to page our on call systems administrator.

The second pattern/action line looks for a */bin/login* syslog message of the form

```
Jul 30 13:49:47 Sierra login:
  REPEATED LOGIN FAILURES ON
  ttyq0 FROM cert.cert.org:
  root, anonyme, anonyme
```

The string REPEATED matches the pattern and swatch executes a script to finger the host that initiated the failed login and store the information for later examination. We are occasionally able to detect compromised accounts with this information.

The rest of the file contains pattern/action lines that are grouped by specific names of machines. It watches out for kernel messages which indicate a potentially serious problem, such as a machine crash or an unexpected reboot. It also looks for messages from the room temperature monitor. When these types of messages are encountered, swatch sends a mail message to our systems administrator mailbox and executes a script to call a pager with a code indicating the system and message type.

Example 2: Individualized Swatch Configuration File

Individuals may design customized swatch configuration files that look for patterns and take appropriate actions depending on their personal preferences. The configuration file shown in Figure 3 is run in a workstation window whenever the system administrator is logged in. The output (a sample of which is shown in Figure 4) is generally ignored or

```
#
# Personal Swatch configuration file to be run in a window on a workstation
#
# These probes should be harmless, but who knows?
#
/fingerd.*(root|[Tt]ip|guest|atkins)/   echo,bell,exec="/bin/date >>
  /home/atkins/tmp/finger.log",exec="/usr/local/etc/backfinger @$6 >>
  /home/atkins/tmp/finger.log"
#
# This should never happen
/su: atkins/                            echo,bell
/su: .* failed/                         echo,bell=3

/[dD]enied/||/DENIED/                   echo=boldunderline,bell

# Alert me of bad login attempts and find out who is on that system
/INVALID|REPEATED|INCOMPLETE/           echo=underline,bell=3

# Important program errors
/LOGIN/                                 echo=boldunderline,bell=3
/passwd/                                echo=bold,bell=3
/ruserok/                               echo=bold,bell=3

# Ignore this stuff
/sendmail/,/nntp/,/xntp|ntpd/,/faxspooler/  ignore

# Report unusual tftp info
/tftpd.*(ncd|kfps|normal exit)/         ignore
/tftpd/                                 echo,bell=3

# Kernel problems
/(panic|halt|SunOS Release)/            echo=blink,bell        3:00     0:16
/file system full/                      echo=bold,bell=3       5:00     0:16

# Try to ignore uninteresting kernel messages
/vmunix.*(at|on)/                       ignore
/vmunix/                                echo,bell              1:00     0:16
```

Figure 3: Personalized swatch configuration file

only occasionally glanced at unless the bell alerts the administrator to a message of interest. Note that the tftpd pattern/action lines in Figure 3 ignore tftp requests from valid hosts and alert the user to invalid requests.

Other Useful Programs

We have written a few scripts which we have found useful when using the swatch package.

```
Sep 13 05:07:23 Sierra vmunix: ie0: no carrier
Sep 13 07:32:07 Sierra ftpd[17910]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 07:35:58 Sierra ftpd[18015]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 07:58:30 gloworm login: INCOMPLETE LOGIN ATTEMPT ON ttyp2 FROM deis17.cin
eca.it
Sep 13 08:15:35 loading-zone.Stanford.EDU vmunix: /loading-zone: file system ful
l
Sep 13 08:15:35 stjames vmunix: NFS write error: on host loading-zone remote fil
e system full
Sep 13 08:53:25 Sierra login: REPEATED LOGIN FAILURES ON ttypb FROM uwmfe.neep.w
isc.edu: help, newuser, d
Sep 13 09:26:59 Sierra su: 'su root' failed for quinn on /dev/ttyp9
Sep 13 09:45:04 espresso.Stanford.EDU login: ROOT LOGIN ttyp0 FROM coffee
Sep 13 10:04:50 Gordon-Biersch vmunix: pid 16100: killed due to swap problems in
exec: I/O error mapping pages
Sep 13 10:05:20 Sierra ftpd[21910]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 10:06:25 Gordon-Biersch vmunix: /tmp: file system full, anon reservation
exceeded
Sep 13 10:06:43 Gordon-Biersch vmunix: pid 16118: killed due to swap problems in
exec: I/O error mapping pages
Sep 13 10:07:02 Gordon-Biersch vmunix: pid 16124: killed due to swap problems in
exec: I/O error mapping pages
Sep 13 10:09:34 Sierra ftpd[22085]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 10:33:55 Gordon-Biersch fingerd[16484]: pudleys.Stanford.EDU (36.2.0.92.1
654) -> "atkins"
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: SunOS Release 4.1.1 (ISL_CLIENT) #
1: Mon Jan 13 08:58:58 PST 1992
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: Copyright (c) 1983-1990, Sun Micro
systems, Inc.
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: mem = 24576K (0x1800000)
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: avail mem = 22630400
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: Ethernet address = 8:0:20:b:67:21
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: cpu = Sun 4/40
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: sd0: <SUN0207 cyl 1254 alt 2 hd 9
sec 36>
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: sd2: <Fujitsu M2624F cyl 1463 alt
2 hd 11 cyl 1463 alt 2 hd 11 sec 63>
Sep 13 11:54:22 espresso.Stanford.EDU vmunix: rebooting...
Sep 13 11:56:40 espresso.Stanford.EDU vmunix: NOT BLOCK: GOTO REQUESTLOOP
Sep 13 11:56:50 espresso.Stanford.EDU vmunix: zs3: silo overflow
Sep 13 12:06:05 Sierra ftpd[28258]: FTP LOGIN FROM vali.Stanford.EDU [36.59.0.32
], fanning
Sep 13 12:11:10 Sierra ftpd[29236]: FTP LOGIN FROM me-bradshaw.Stanford.EDU [36.
65.0.71], bradshaw
```

Figure 4: Output from swatch using the configuration file in Figure 3 over the course of 6 hours and more than 2300 lines of input

Reswatch

Reswatch was written to run out of cron periodically. It finds all instances of swatch that the user is running and sends a SIGHUP. This is useful if swatch is getting its input from an active log file, like syslog, that is moved and rendered inactive. Since we want to start getting our input from the new active log file, the old file handle needs to be closed and the new one opened. This effect is achieved when swatch aborts one script and starts a new one after receiving a SIGHUP.

Backfinger

Backfinger is used to finger the host that generated an unsuccessful login attempt. Output from this command is placed in its own log file. Backfinger uses safe_finger to filter out potentially dangerous output from remote finger servers. This is most useful when culprits fail to log in to a system using an unauthorized account, like root, guest, or anonymous. Some administrators might be surprised at how often this happens on their systems.

CallPager

For those who must carry a pager, this is very useful for receiving urgent information, such as serious system failures or possible security breaches. This is a simple script which uses the UNIX tip command to call a pager through a modem and leave a code number to indicate the type of message detected. Users can customize the codes so that they can tell exactly what type of message was detected, and the system from which it came.

Conclusions

Over the past year and a half swatch has proven to be a valuable tool for monitoring the health of a large collection of workstations and servers. On several occasions we have been able to detect intruders probing our systems who would probably have been missed without centralized logging and swatch. On a few occasions it prevented system meltdown when air conditioning units failed on a weekend or late at night. Its value has increased as we have gathered more experience in optimizing the swatch configuration file entries.

In the near term, we see a need to improve the logging capabilities of additional system utilities (i.e. sendmail, ntp, ypserv, xdm, xlogin). We plan to gather suggestion from other sites using the package before making substantial changes to swatch itself.

Availability

Swatch source and documentation along with its companion scripts are available via anonymous ftp from Sierra.Stanford.EDU, [36.2.0.98], in the pub/sources directory. Listserver access is available from listserver@Sierra.Stanford.EDU.

Author Information

Stephen E. Hansen received the B.S. and M.S. degrees in Electrical Engineering from Stanford University in 1976 and 1981 respectively. In 1975 he joined the Integrated Circuits Laboratory at Stanford University, first as Systems Programmer, and since 1978 as Senior Scientific Programmer. In 1983 he organized the Electrical Engineering Computer Facility at Stanford where he currently serves as its Director. Mr. Hansen can be reached via U.S. Mail at the Applied Electronics Laboratory 218, Stanford, CA 94305-4055 or via electronic mail at hansen@sierra.stanford.edu.

Todd Atkins received a B.S in Electrical Engineering from Stanford University in 1988. Since 1987 he has been with the Electrical Engineering Computer Facility as a Systems Administrator. Mr. Atkins can be reached via U.S. Mail at the Applied Electronics Laboratory, Room 113, Stanford, CA 94305-4055 or via electronic mail at Todd_Atkins@eecf.stanford.edu.

References

[1] W. Venema. "TCP WRAPPER, A Tool for Network Monitoring, Access Control, and for Setting Up Booby Traps", Proc. 1992 USENIX Security Symposium, USENIX Association, Sept. 1992.

[2] L. Wall and R. Schwatz. "Programming Perl", O'Reilly and Associates, Sebastopol, CA. 1991.

Appendix A: A Syslog Configuration File.

```
# syslog configuration file.
#
# Master syslog configuration file.
#
# This file is processed by m4 so be careful to quote ('') names
# that match m4 reserved words. Also, within ifdef's, arguments
# containing commas must be quoted.
#
# Note: Have to exclude user from most lines so that user.alert
#       and user.emerg are not included, because old sendmails
#       will generate them for debugging information. If you
#       have no 4.2BSD based systems doing network logging, you
#       can remove all the special cases for "user" logging.
#
*.err;kern.debug;auth.notice;user.none                      /dev/console
*.err;kern.debug;daemon,auth.notice;mail.crit;user.none     /var/adm/messages
lpr.debug                                                   /var/adm/lpd-errs

# You may want to add operator to the following if your operator
# is a traditional Unix style operator.
*.alert;kern.err;daemon.err;user.none                       root
*.emerg;user.none                                           *

ifdef('LOGHOST',
# for loghost machines, to have authentication messages (su, login, etc.)
# logged to a file, un-comment out the following line and adjust the file
# name as appropriate.
auth.notice                                                 /var/log/authlog
daemon.info;auth.notice;mail.debug;kern.debug               /var/log/syslog
*.err;daemon.none;mail.none;kern.none;auth.none;user.none /var/log/syslog
)

# following line for compatibility with old sendmails. they will send
# messages with no facility code, which will be turned into "user" messages
# by the local syslog daemon. only the "loghost" machine needs the following
# line, to cause these old sendmail log messages to be logged in the
# mail syslog file.
#
ifdef('LOGHOST',
user.alert                                                  /var/log/syslog
)

#
# non-loghost machines will use the following lines to cause "user"
# log messages to be logged locally.
#
ifdef('LOGHOST', ,
user.err                                                    /dev/console
user.alert                                                  root
)

# Send most everything to the LogMaster. If this is the logmaster,
comment out the following two lines
*.info;kern.none;mail.none                                  @logmaster
kern.debug;mail.err                                         @logmaster
```

NAME

swatchrc – configuration file for the simple watcher swatch(8)

SYNOPSIS

~/.swatchrc

DESCRIPTION

This configuration file is used by the **swatch(8)** program to determine what types of expression patterns to look for and what type of action(s) should be taken when a pattern is matched.

The file contains four TAB separated fields:

```
/pattern/[,/pattern/,...]     action[,action,...] [[HH:]MM:]SS   start:length
```

A pattern must be a regular expression which **perl(1)** will accept, which is very similar to the regular expressions which **egrep(1)** accepts.

The following actions are acceptable:

echo[=mode]	Echo the matched line. The text mode may be *normal, bold, underscore, blink, inverse.* Some modes may not work on some terminals. **Normal** is the default.
bell[=N]	Echo the matched line, and send a bell *N* times (default = 1).
exec=command	Execute *command.* The *command* may contain variables which are substituted with fields from the matched line. A *$N* will be replaced by the *Nth* field in the line. A *$0* or *$** will be replaced by the entire line.
ignore	Ignore the matched line.
mail[=address:address:...]	Send *mail* to *address(es)* containing the matched lines as they appear (default address is the user who is running the program).
pipe=command	Pipe matched lines into *command.*
write[=user:user:...]	Use **write(1)** to send matched lines to *user(s).*

The **third** field (which is optional) can contain a time interval. The time should be in one of three formats:

SS	-- *Just seconds*
MM:SS	-- *Minutes and seconds*
HH:MM:SS	-- *Hours minutes and seconds*

If an interval is specified and more than one identical line is received, **swatch** will not do the actions specified until the specified time has elapsed. If the action that is performed uses the input line then the number of lines will be included in the line.

The **fourth** field is also optional and must only exist if the the third field exists. This field is used to specify the location of the time stamp in the log message as well as the length of the time stamp. It should be specified in the form start:length.

EXAMPLE

```
/file system full/ echo,bell        01:00   0:16
```

This example a line which contains the string "file system full" will be echoed and the screen bell will sound. Also, multiple instances of the message will not be echoed if they appear within a minute of the first one. Instead the following message will be acted upon after the time interval has expired. This is what may appear if a the message appeared 20 times.

```
*** The following was seen 20 times in the last 1 minute(s):
        ==> EE-CF.Stanford.EDU: /var: file system full
```

SEE ALSO

swatch(8), **perl**(1), **egrep**(1),

AUTHOR

E. Todd Atkins (Todd_Atkins@EE-CF.Stanford.EDU)
EE Computer Facility
Stanford University

NAME

swatch – simple watcher

SYNOPSIS

swatch [**–c** *config_file*] [**–r** *restart_time*] [**–P** *pattern_separator*] [**–A** *action_separator*] [**–I** *input_record_separator*] [[**–f** *file_to_examine*] | [**–p** *program_to_pipe_from*] | [**–t** *file_to_tail*]]

DESCRIPTION

Swatch is designed to monitor system activity. **Swatch** requires a configuration file which contains *pattern(s)* to look for and *action(s)* to do when each pattern is found.

OPTIONS

–c *filename* Use *filename* as the configuration file.

–r *restart_time* Automatically restart at specified time. *Time* can be in any of the following formats:

+hh:mm

Restart after the specified time where *hh* is hours and *mm* is minutes.

hh:mm[am|pm]

Restart at the specified time.

–P *pattern_separator*

Tells **swatch(8)** to use *pattern_separator* when parsing the patterns in the configuration file. The default is a comma.

–A *action_separator*

Tells **swatch(8)** to use *action_separator* when parsing the actions in the configuration file. The default is a comma.

–I *input_record_separator*

Tells **swatch(8)** to use *input_record_separator* as the character(s) which mark the boundary of each input record. The default is a carriage return.

You may specify only one of the following options:

-f *filename*	Use *filename* as the file to examine. **Swatch** will do a single pass through the named file.
-p *program_name*	Examine input piped in from the *program_name*.
-t *filename*	Examine lines of text as they are added to *filename*.

If swatch is called with no options, it is the same as typing the command line

```
swatch -c ~/.swatchrc -t /var/log/syslog
```

SEE ALSO

swatch(5), **signal**(3)

FILES

/var/tmp/..swatch..PID	Temporary execution file

AUTHOR

E. Todd Atkins (Todd_Atkins@EE-CF.Stanford.EDU)
EE Computer Facility
Stanford University

NOTES

Upon receiving a ALRM or HUP signal swatch will re-read the configuration file and restart. Swatch will terminate gracefully when it receives a QUIT, TERM, or INT signal.

Where Did All The Bytes Go?

Dinah McNutt – Tivoli Systems
Michael Pearlman – Rice University

ABSTRACT

Configuring, installing, re-configuring, and re-installing disk drives can be time consuming. Understanding the physical disk and how UNIX file systems are laid out on disk drives can not only help an administrator troubleshoot problems, but can allow him/her to maximize the amount of disk space available. Standard formatting utilities tend to configure disks with similar geometry identically. By using different geometry parameters or third party formatting programs, you can get the maximum amount of disk space from your drives.

This paper is a tutorial on understanding SCSI disk drive geometry (and how it differs from more traditional drives) and the steps required to get a disk drive to the point where it is usable (e.g., with a UNIX file system on it.) It includes practical tips such as questions to ask vendors so you know how much usable space you will have when you purchase a disk drive. It also include a case study where we show how you can squeeze additional usable disk space from a drive. This paper is targeted at novice system administrators, but experienced administrators who want to learn more about disk drives will hopefully learn something from reading this paper. The examples we use will be based on a BSD system, but the concepts apply to other types of UNIX systems as well.

Background and Motivation

Disk drives are probably one of the biggest headaches administrators will ever run across. Our experience has been that the number of disk drives increases the support effort significantly. One of the reasons is the amount of time involved in installation and reconfiguration of disk drives. Figuring out how to configure the drives and backing up and restoring the files on the existing drives can take a lot of time.

Standard hardware interfaces (de facto as well as official) have helped create a market where users can pick and choose different combinations of disk drives and controllers. The Small Computer System Interface (SCSI) is the most common type of bus found on workstations today. Vendors typically use a proprietary, high-speed bus to connect CPUs with memory and SCSI to connect I/O devices and peripherals. The SCSI specification is published and vendors have used this specification to manufacture hundreds of SCSI devices. This allows customers to select the device that best meets their needs and the computer vendors can focus on their processor and operating-system technologies.

Many computer vendors supply default configuration information for configuring disks drives. However, these defaults are not optimized for each disk and by taking the time to perform the calculations yourself, you can maximize the amount of disk space available after configuring the disk.

UNIX Disk Hierarchy

UNIX presents the user with hierarchical view of a disk. At the bottom level is the physical disk as seen by the device driver. At this level, a SCSI disk appears as an array of sectors and all commands to the disk must be SCSI commands. The next level is the raw disk where the disk appears as an array of sectors but can be accessed via read, write and ioctl system calls. The raw disk is used when high speed is required without the overhead of a file system structure (e.g., backup software.) The top level is the UNIX file system which presents the user with the tree structured directory system with which we are all familiar. At this level, the disk interface appears as an array of units called blocks. The file system is accessed as blocks and subdivisions of blocks called fragments. Each block is a multiple of the disk size and can only be divided into 2, 4 or 8 pieces. The file system block and fragment size are defined at file system creation time. Typical values are 8192 byte block size paired with a 1024 byte fragment size or 4096 byte block size paired with a 512 byte fragment size.

Each level of the hierarchy uses some of the available disk space to implement its own level of abstraction. Therefore, the usable capacity of a given disk depends on the level of the hierarchy at which it is being used: you have the largest capacity at the physical level and the least at the file system level. The operating system often reserves some space to store alternate disk labels and other essential information which is not accessible above the physical disk level. The file system reserves space for super blocks, cylinder group information and for inodes.

Units of Capacity

In the early days, disks had small capacity and were terribly expensive. Vendors quoted disk storage in small units (e.g., kilobytes) and there was little confusion. Today disks are large and the cost has fallen dramatically. All other things being equal, the disk that you should purchase is the one that gives the best price per unit of storage. At some point in time, a vendor came up with the clever idea of redefining the unit of capacity to be smaller to make his/her company's disks appear more cost effective. Other companies followed suit and all disk sizes were quoted in terms of the smaller unit. Today we have two different numbers associated with each of the terms megabyte and gigabyte which has led to the anomaly that one megabyte of memory does not fit in one megabyte of disk. For the purposes of this paper we will use the following terms:

Term	Size in Bytes
1 megabyte (Mbyte)	2^{20} or 1,048,576
1 disk megabyte (DMbyte)	1,000,000
1 gigabyte (Gbyte)	2^{30} or 1,073,741,824
1 disk gigabyte (Dgbyte)	1,000,000,000

When the actual unit is unknown, we shall use ?MByte and ?Gbyte.

Rounding

Disk capacities are rounded numbers. A 1.6 Dgigabyte disk can really be as small as a 1.55 Dgigabyte disk. The loss of 0.05 Dgigabytes may look small but it is 50 Dmegabytes. With the current increasing trends in disk sizes, this rounding will be even larger when the Dterabyte disks arrive. To protect yourself, be sure to ask for your quotes in megabytes, kilobytes or even bytes. A reputable dealer has these figures available and should quote you a number plus or minus a small amount to account for the number of defects on a drive.

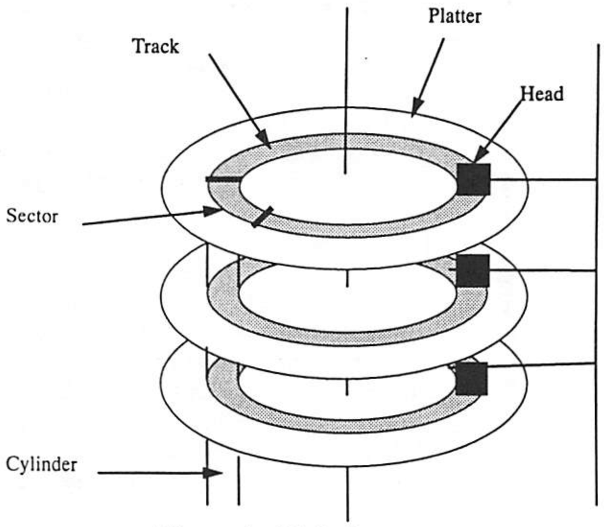

Figure 1: Disk Geometry

Disk Geometry

A disk drive is composed of one or more circular platters. Most platters can be written on both sides and the drive has as many recording heads as it does usable surfaces. The heads move radially across the platters and may be connected independently or grouped on an armature. Each radial location is called a track and each track is divided into smaller sections called sectors as show in Figure 1. A group of tracks located at the same radial position from the center of the platter is called a cylinder.

The heads float a microscopic distance off the platters. A head crash occurs when a head touches the surface of the media, removing some of the media and damaging the head. This usually results in an unusable disk drive.

Modern disks have either a fixed or variable geometry. Fixed geometry means that the number of sectors per track is constant throughout the entire drive. This is the geometry that was assumed in the design of many of the file systems found on today's machines. Variable geometry means that the number of sectors per track is not constant throughout the drive. The normal pattern for a variable geometry is that the number of sectors per track is smallest on the innermost tracks and increases as the radius of the track increases. The most common example of variable geometry is found in disks that use zone bit recording where the disk is divided into sets of contiguous cylinders, called zones, with a constant number sectors per track in each zone. The sector per track number increases as the distance of the zone from the disk center increases. This method increases the number of sectors you can have on the disk since you no longer are limited by the number of sectors that can be squeezed on the innermost cylinder.

Making the Disk Usable

The steps involved in making a disk drive usable are formatting, partitioning and labeling, and optionally creating file systems. The following sections will look at each of these steps in detail.

Formatting

Formatting is the process by which address information and timing marks are written on the disk to define each sector. The format process has two levels. The lowest level consists of

- Defining the disk geometry
- Selecting the type of defect management
- Defining sector layout parameters
- Enabling optional drive features
- Laying out the disk sectors and sector headers

Low-level Formatting

Low-level formatting is accomplished by the host computer sending the appropriate SCSI commands to the drive. The top level consists of the host writing operating system specific information to the disk. Typical information includes at a minimum a label which contains the disk geometry parameters used to format the drive. To view the label information on a SunOS 4.1.3 system, use *dkinfo* (8[1])

```
# dkinfo sdl
1307 cylinders 9 heads 70 sectors/track
a: 32130 sectors (51 cyls)
   starting cylinder 0
b: 59850 sectors (95 cyls)
   starting cylinder 51
c: 823410 sectors (1307 cyls)
   starting cylinder 0
d: No such device or address
e: No such device or address
f: No such device or address
g: 731430 sectors (1161 cyls)
   starting cylinder 146
h: No such device or address
```

The format program gets the required information either from a static table stored in a system file or by an interactive interrogation of the user. Typical information that is required is

- The number of accessible cylinders
- The number of data heads
- The number of sectors per track
- The sector size
- The rotational speed of the disk
- Type of defect management and other related parameters

Obtain this information by reading the appropriate disk drive manual and trial and error. Formatting with the wrong choice of parameters can render a disk useless and in a state that requires it be returned to the vendor. So, we recommend obtaining the information from your vendor at time of purchase.

Most of the information required by formatting programs is not necessary for SCSI drives. SCSI disks are intelligent. The SCSI drive can be sent the desired sector size and the defect handling parameters and then can be commanded to format itself. When the format has completed, the host computer can request the drive to return its capacity in sectors as well as its head count, cylinder count and average sectors per track count which can be used for labeling and partitioning the disk. There are third party vendors marketing inexpensive disk utilities for SCSI disks which include a formatting program based upon the scheme outlined above.

Choosing the Sector Size

In this section, we shall discuss the effect of sector size on the "formatted" capacity of a disk drive. On most systems, the sector size is fixed by the operating system and the user has no freedom to choose the size without rewriting parts of the file system code. On UNIX systems, this value has traditionally been 512 bytes. In addition to the 512 bytes of data storage space, there is disk and controller overhead for storing such information as the sector ID. This overhead can occupy around 100 additional bytes which is approximately 16% of the total sector.

Therefore, by using a larger the sector size, there is less space wasted on headers and more space available for data storage. For example, the CDC Wren VII disk has an unformatted capacity of 1200 ?Mbytes[2] and with no sectors allocated for defect handling has a formatted capacity of 1050 ?Mbytes at 512 bytes per sector and 1106 ?Mbytes at 1024 bytes per sector.

Defect Management

Disk media is subject to imperfections. These imperfections may be due to the manufacturing process or to normal wear and tear. Mechanisms must be in place to ensure that these defective locations are not used to store data. The manufacturer of a SCSI disk uses sophisticated equipment to test each disk and stores on the drive a list of known defective locations. This list may grow automatically as the disk detects and optionally repairs new defects or may be manually supplemented. Defect management strategies fall in two camps:

1) Make the host computer responsible for avoiding the defective locations. The system can use the defect list as a map of the known bad sectors on the disk. When a bad sector is the next to be allocated, the device driver can switch to a free sector on the same track, skipping the bad sector.
2) Allocate space sectors during formatting which will not be visible to the device driver but which the disk can use to replace the defects.

The strategy chosen can have a dramatic effect on performance and formatted capacity. We shall discuss the following defect management strategies:

1) no management
2) host-based management
3) track-based sparing
4) cylinder-based sparing

The no management strategy formats the disk with no sectors allocated to handle defects and totally ignores the defect list on the disk. This was a common strategy many years ago. A new disk was formatted and a hard copy of the defect list was obtained using one of the following methods:

- Using the hardcopy defect list shipped with the drive
- Waiting for bad sectors to show up
- Running a program that does a read-write-read to every sector on the disk and noting the sectors that caused errors

[1]The authors have reformatted and deleted extraneous information from command output throughout this paper.

[2]It is the authors' belief that ?Mbytes figures in this paragraph are actually DMbytes.

The user would have to manually hand edit the inode and free list to indicate that these sectors were allocated to a file with a special name. These systems had modified utilities to never touch files with the magic name. Programs that accessed raw disk space were modified not to touch the bad sectors. This scheme maximized the formatted capacity but did not work if critical system blocks were defective and relied on the user never accessing the *bad* file. This scheme is rarely used today.

The host-based management strategy is to also format the disk without allocating any spare sectors but to have the device driver note the address of defective sectors. The device driver then must guarantee that the defective sectors are never accessed. This also maximizes capacity but places a lot of complexity into the device driver code.

Track-based sparing allocates a fixed number of spare sectors per track and spare tracks per disk. We'll refer to the number of spare sectors per track as *asect* and it is usually 1. We'll refer to the number of spare tracks per volume as *atrks* and it is usually twice the number of heads on the drive. If the number of defects on a particular track is less than or equal to *asect* then no seek or head switch is required by the drive to use the spare sector instead of the defective one. If there are more than *asect* defects on a particular track then this entire track is replaced by one of the spares and a seek will occur and possibly a head switch. This scheme gives the highest performance but the lowest capacity.

Cylinder-based sparing allocates *asect* spare sectors per cylinder and *atrks* spare tracks per volume which is chosen to be an integral number of cylinders (e.g., no cylinder fractions.) If there are less than or equal to *asect* defects in a given cylinder then a defective sector is replaced by one of the spares allocated in the same cylinder, which at worst results in a head switch. If the number of defects in a given cylinder exceeds *asect*, then some of the tracks will be mapped to one of the spare tracks. This will cause a seek to occur and possibly a head switch. If *asect* is less than the number of disk heads, this scheme will give a larger formatted capacity than track-based sparing with only a slight performance hit.

For SCSI disks, the best strategies to use are either track-based or cylinder-based sparing. The best way that we have found to tune the parameters in track or cylinder-based sparing is to format the drive and extract the defect list. By looking at the distribution of defects, the values of *asect* and *atrks* can be adjusted to optimal values and the disk reformatted with these values. Remember to allow some space for future defects.

Partitioning and labeling

A partition is a contiguous number of sectors that will be accessed as a raw disk device. You can have up to 8 partitions on a disk and the partitions can be defined anyway you wish. The partitions are indicated by an alphabetic character (a-h) and, traditionally, partition c is the whole disk. Figure 2 shows a disk that has been partitioned into 4 partitions: a, b, c, and g. Notices that you may use either a,b, and g or you may use c, but you may not use all 4 partitions as data written to partition c will over-write data on partitions a and b and vice versa.

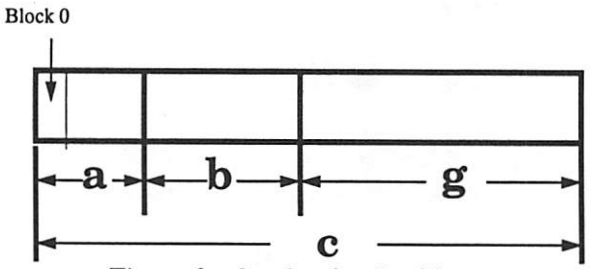

Figure 2: Overlapping Partitions

A file system is created using a disk partition. A file system partition may be as large as the whole disk or as small as a single cylinder. In all cases, the file system should start and end on a cylinder boundary as shown in Figure 3. Some versions of UNIX expect the file system to end on a cylinder boundary and you may get unexpected results if you are not careful about how you lay out your file systems.

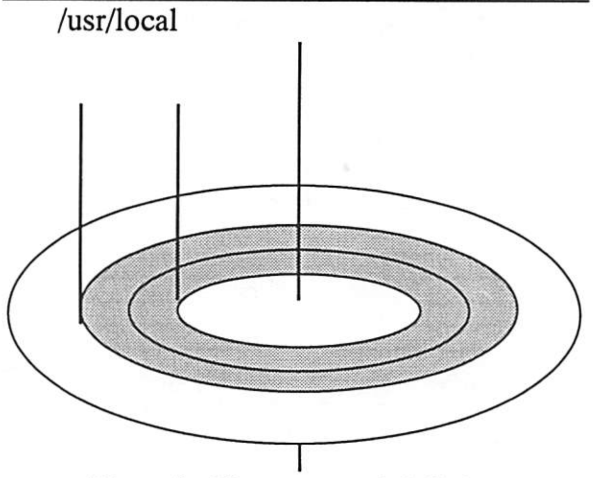

Figure 3: Filesystems and Cylinders

The Berkeley Fast File System is structured as a set of contiguous cylinder groups. For fast file system partitions it is important that the partition size be a multiple of the number of sectors in a cylinder group so that no space is wasted. Any cylinders that are left over due to this constraint can be assigned to a swap partition or to any other partition which will be accessed solely as a raw disk.

The disk label contains the information about how many partitions are on the disk and where they are located. The label is normally located in block 0 of the disk. As shown in Figure 2, block 0 is also part of partition a. It is a good idea not to make the first partition on your disk a swap partition as some version of UNIX will allow you to swap on block 0, thus overwriting your disk label. Some versions of UNIX store the label in a protected partition, so you cannot accidentally over-write the label. Check with your vendor to see how your system(s) work.

Making file systems

A file system is composed of superblocks, inodes and data blocks. The superblock stores information such as the size of the file system, pointers to the inodes, etc. Inodes are where the information about each file is contained: the size of file, last date modified, access permissions, user owner, group owner and pointers to the data blocks allocated to the file.

We will limit our discussion of making file systems to some of the issues that impact capacity. On BSD systems, the *newfs* command is used to create a file system on a raw disk partition. The options to *newfs* of interest are:

- i This options specifies the number of bytes/inode or the density of inodes in the file system. On SunOS 4.1.3 systems, the default is 8192 bytes/inode. For most file systems, you can reduce this number to 4096 bytes/inode with no problems. The exception is file systems that have many small files. By using 4096, you decrease the total number of inodes on the file system and increase the amount of space available for data storage.
- m The minimum free space threshold which is reserved from normal users. This value is usually 10%. One a 1 Gbyte system, 10 % is around 100 Mbytes which is a significant amount of space. On large disks, we recommend reducing this value to 5 %. BSD systems have a command called *tunefs* which allows you to modify the minimum free space threshold after the file system has been created.
- o File systems can be optimized to either minimize the amount of time spent allocating blocks or minimize the space fragmentation on the disk. By default, if the minimum free space threshold is less than 10%, the file system is optimized for space. If it is 10% or greater, it is optimized for time. Some implementations dynamically change the optimization based on the amount of free space in the file system.

Newfs is simply an easy-to-use front end for *mkfs* which is found on both BSD and System V systems (although the command is different on each system.)

Determining Usable Capacity

Unformatted versus formatted capacity

Disk specification sheets cite either unformatted or formatted capacity. Examining these numbers to determine how much data space you are buying requires expertise and should not be necessary. However, dealers cite these numbers and you need to know what they mean.

The unformatted capacity is the number of raw bytes on the disk and will always be more than the formatted value. The formatted value is the manufacturer's estimated at how much space will be available for user data after the disk has been formatted. This number will vary on different UNIX systems, so ask the vendor what the formatted capacity will be on your system.

Case Study - SunOS and Maxtor LX213S

Recently, we had to replace one of our Sun 207 Dmegabyte internal SCSI disk drives. This drive is listed by Maxtor as having 248 Dmegabytes unformatted capacity and 213 Dmegabytes of formatted capacity. We will look at 3 different methods of formatting the disk drive and see which method yields the maximum usable disk capacity.

The first example uses Sun's format program with the values supplied from /etc/format.dat:

```
disk_type = "SUN0207" \
    : ctlr = SCSI \
    : trks_zone = 9 : atrks = 2 \
    : asect = 4 : ncyl = 1254 \
    : acyl = 2 : pcyl = 1272 \
    : nhead = 9 : nsect = 36 \
    : rpm = 3600 : bpt = 18432
```

Using *dkinfo* to display the number of available sectors:

```
# dkinfo sd0
1254 cylinders 9 heads 36 sectors/track
c: 406296 sectors (1254 cyls)
```

We see that this disk has a real formatted capacity of 406296 * 512 = 208023552 bytes or 208 Dmegabytes. Observe that there is a 5 Dmegabyte discrepancy between our formatted capacity and Maxtor's quoted formatted capacity. Now, let's see how much usable capacity is available after making the UNIX file system.

```
# newfs /dev/rsd0c
/dev/rsd0c: 406296 sectors
            in 1254 cylinders
            of 9 tracks, 36 sectors
    208.0MB in 79 cyl groups
    (16 c/g, 2.65MB/g, 1216 i/g)
```

```
# df /dev/rsd0c
Filesystem    kbytes    used   avail
/dev/sd0c     189858       9  170863
```

```
# df -i /dev/rsd0c
Filesystem     iused    ifree
/dev/sd0c          4    96060
```

Creating the file system cost 208023552 - 189858*1024 = 13608960 or 6.5 percent of the formatted capacity. There is an additional 10 percent of the data space that is held back from normal users as expected. Now recreate the file system with approximately 50 percent less inodes:

```
# newfs -i 4096 /dev/rsd0c
/dev/rsd0c: 406296 sectors
            in 1254 cylinders
            of 9 tracks, 36 sectors
   208.0MB in 79 cyl groups
   (16 c/g, 2.65MB/g, 640 i/g)
```

```
# df /dev/rsd0c
Filesystem    kbytes     used   avail
/dev/sd0c     195546        9  175982
```

```
# df -i /dev/rsd0c
Filesystem     iused    ifree
/dev/sd0c          4    50556
```

Now file system creation cost 208023552 - 195546*1024 = 7784448 bytes or 3.7 percent of the formatted capacity. Again, we see that 10 percent of the data space was held back.

The second example uses a commercial formatting program which obtains the disk geometry from the disk.

```
# dkinfo sd0
1411 cylinders 7 heads 42 sectors/track
c: 414834 sectors (1411 cyls)
```

Notice that the disk geometry is totally different from the physical geometry of the disk which is used by Sun's format which is why the disk appears to have only 7 heads. The disk's formatted capacity increased to 414834*512 = 212395008 bytes or 212.4 Dmegabytes. In this case, there is only 0.6 Dmegabytes difference from Maxtor's formatted capacity figure. Moreover, since the SunOS reserves 2 cylinders for defect management, adding this back into our formatted capacity we get 212395008 + 2*7*42*512 = 212696064 bytes or 212.7 Dmegabytes which with rounding is equal to Maxtor's figure. Let's look at the usable data capacity. We'll create the file system using a reduced number of inodes since we have already demonstrated this will increase the amount of usable disk space.

```
# newfs -i 4096 /dev/rsd0c
/dev/rsd0c: 414834 sectors
            in 1411 cylinders
            of 7 tracks, 42 sectors
   212.4MB in 89 cyl groups
   (16 c/g, 2.41MB/g, 576 i/g)
```

```
# df /dev/rsd0c
Filesystem    kbytes     used   avail
/dev/sd0c     199567        9  179601
```

```
# df -i /dev/rsd0c
Filesystem     iused    ifree
/dev/sd0c          4    51260
```

We now see a loss of 212395008 - 199567*1024 = 8038400 or 3.8 percent of the formatted capacity.

By not using Sun's format program and the default number of inodes we gain 9942016 bytes (9.9 Dmegabytes) of data space or 4.7 percent of the formatted capacity quoted by the vendor. Both of the format programs in this example use cylinder-based defect handling but the Sun program used four spare sectors per cylinder and the third party program only three. It has been our experience that this third party program shows even more impressive gains on larger disks with more heads.

Summary

Vendors cite many different numbers as the capacity of a given disk. It is important when comparing disks to use the same metrics. We hope that this paper will help you to understand these metrics. We have given some examples of parameters that can be changed with the goal of increasing the data space of a given disk.

Some caveats are in order. Formatting is a dangerous process and has the capability of rendering a disk useless. The best choice for the partition size and file system parameters depend on the disk usage patterns. These parameters are used to determine static data structure sizes which if too small can inhibit further data storage even though space is available. Poorly chosen values have the ability to lessen disk performance when space becomes tight.

Acknowledgements

Many thanks to Tom Skerl of Applied Tensor Technology for his helpful comments and to Britton Dick of Rare Systems for his patience in supplying disk parameters.

References

Applied Tensor Technology, *SCSI Disk Utilities User's Guide*, Version 1.7, January 1993.

Control Data Corporation, *Product Specification For The Wren VII SCSI Disk Drive Model 94601.*

P. Galvin, D. McNutt, and M. Pearlman, "Tricks Of The Trade For Systems Administrators", *Sun User Group Proceedings*, December, 1992.

M. Loukides, *System Performance Tuning*, O'Reilly and Associates, 1991.

Maxtor Corporation, *XT-8000S Product Specification and OEM Technical Manual.*

D. McNutt, "Where Did All The Bytes Go?", *SunExpert Magazine*, pp. 49-53, May 1991.

D. McNutt, "SMD Disk Drives: PartII", *SunExpert Magazine*, pp. 44-47, June 1991.

Author Information

Michael Pearlman is the System Manager for the Department of Computational and Applied Mathematics and for the Department of Statistics at Rice University. Reach him via U. S. Mail at Department of Computational and Applied Mathematics; Rice University; POB 1892; Houston, TX 77251. Reach him electronically at canuck@rice.edu.

Dinah McNutt is a System Administration Consultant for Tivoli Systems where she works with customers of Tivoli helping them with system administration problems and customization of Tivoli's system administration software. She has been doing system administration for over 8 years and has written technical articles on the subject for SunExpert Magazine, RS/Magazine, and the X Resource Journal. Ms. McNutt currently writes the Daemons and Dragons column for UNIX Review magazine. She can be reached at dinah@tivoli.com.

A Practical Approach to NFS Response Time Monitoring

Gary L. Schaps and Peter Bishop – Cirrus Logic, Inc.

ABSTRACT

NFS and the automounter simplify both filesystem access and filesystem management in large, distributed networks. In demanding environments, however, they require a network management strategy which ensures a consistent and acceptable level of performance.

This paper reviews available tools for measuring NFS response time including rpcspy, nfswatch, nfsstat, and WireTap. We then describe a heuristic technique we have implemented on top of basic NFS response time statistics. It is used to create a management measure of network quality based upon NFS response time, and to proactively focus on NFS performance bottlenecks.

Introduction

In a time-to-market driven environment where everyone's success depends upon networked computing resources, users expect transparent, reliable and fast access to NFS filesystems. While the automounter eases the administrative overhead of providing transparent and reliable access, managing NFS performance requires some additional tools. Why monitor NFS response time? Consider the impact of a change in average NFS response time on the order of only one millisecond per NFS operation. Is it significant? The answer is yes. A busy filesystem can service hundreds of thousands of NFS operations in a short time, and a small change in the response time of each operation quickly adds up. A network which exhibits poor or inconsistent response time wastes users time and ultimately hurts the performance of the organization. Often the cause may be relatively easy to correct, if detected. However, if no systematic effort is made to manage response time, bottlenecks will be overlooked until users complain that "the network is slow".

We have undertaken the task of designing and implementing a continuous improvement program based upon NFS response time. Our initial objective was to produce a management measure of service quality for each of our subnets. We chose NFS response time because it represents the best single measure of overall network performance. Our plan was to establish the measure, track it each month, and use it as a baseline for efforts at continuous improvement. This approach suggests a somewhat general view of NFS response time, but in fact we use a "bottom up" approach designed to enable us to detect the signatures of specific system and network problems and to measure our success at dealing with them. We routinely gather the most detailed data we can collect on all our subnets and analyze it with tools developed in perl, creating

1) a single NFS response time "figure of merit" for each subnet,
2) a list of filesystems which exhibit "poor" performance, and
3) a detailed response time profile for each filesystem partitioned by NFS operation category (read, write, etc.) and performance threshold (excellent, good, etc.).

Collecting NFS Response Time Data

A large network, comprised of many subnets, presents a challenging environment for reliable, automated network monitoring. An often cited design criteria is to avoid a centralized monitoring and data collection point [Lehman92]. We agreed that a distributed system was desirable and began to look at existing tools for collecting NFS response time data. Some earlier work [Keith90, Shein89, Watson92] used nfsstat statistics or "similar kernel meters" to characterize NFS file server performance. With "nfsstat -m" one can track smoothed round trip time for NFS Lookup, Read, and Write operation categories on all mounted filesystems (see Figure 1).

This is close to what we wanted, and an earlier version of our NFS response time monitor used nfsstat. The trouble with nfsstat is that it groups NFS operations into predefined categories and deprives one of the ability to create categories based upon locally observed operation mixes and performance. In addition, getting a network perspective on NFS response time using nfsstat implies a lot of overhead – running it on each client and collecting the results over the network. Lastly, nfsstat presents a slightly confusing picture of what filesystems are actually mounted and what server exports them when the automounter is in use.

Another useful tool for monitoring NFS response time is "nfswatch" [Curry93]. Unlike nfsstat, it is a passive network monitoring tool which observes all NFS traffic on a network and logs useful statistics including the frequency of each type of NFS operation and its average response time. While frequency is reported by filesystem, average response time is not. Passive network monitoring virtues include providing a network view of NFS response time and minimizing the runtime overhead associated with acquiring that view (see Figure 2).

Passive network monitoring is also the approach used by rpcspy [Blaze92] which produces a record for each transaction containing a timestamp, server name, client name, the length of time the NFS procedure took to execute, the name of the procedure, command specific arguments and return data. One of the arguments is an NFS filehandle within which is embedded enough information to determine which of the servers' filesystems is involved in the procedure. Computing the average response time for each NFS procedure by server and filesystem can be accomplished by analyzing the output of rpcspy.

A commercial software tool for monitoring NFS response time is WireTap [AIM92]. It also uses a distributed, passive network monitoring architecture and creates near real-time graphical display of NFS response time as well "expert alarms" which suggest causes and remedies for performance problems. In addition to NFS response time, WireTap also represents NFS load (operations per second), NFS retransmits, a derived metric called "file server efficiency" and Ethernet wire loading over both short and long term time intervals. No filesystem or individual NFS operation level detail is available from WireTap.

We initially used nfswatch to collect NFS response time data from our networks and characterized the local latencies of each NFS operation type. Subsequently, we changed the format of rpcspy's output to include frequency, average response time, and worst case response time for each NFS operation by filesystem. We now run this modified rpcspy during the first thirteen days of each month on all our subnets, averaging response times in twenty minute intervals and collecting the data on a single NFS filesystem for analysis.

Analyzing NFS Response Time Data

NFS defines 18 remote procedure call (RPC) operations, most of which are analogous to Unix system calls. These include operations to read and write a file (read, write), obtain a files attributes (getattr), obtain the contents of directories (lookup,

```
Server (pid104@/news),   (Addr 127.0.0.1)
Flags: hard int  read size=8192, write size=512,  count = 5
Lookups: srtt=2 (5ms), dev=2 (10ms), cur=1 (20ms)
Reads: srtt=0 (0ms), dev=0 (0ms), cur=0 (0ms)
Writes: srtt=0 (0ms), dev=0 (0ms), cur=0 (0ms)
All: srtt=2 (5ms), dev=4432 (22160ms), cur=2216 (44320ms)
```

Figure 1: Output from "nfsstat -m" (srtt = smoothed round trip time)

```
Procedure           int   pct    total  completed  ave.resp  var.resp  max.resp
 CREATE              16    0%       75         16     69.02   5182.62    262.05
 GETATTR            925   26%     4029        925     13.45    362.51    189.72
 GETROOT              0    0%        0
 LINK                 0    0%        0
 LOOKUP             759   21%     3327        759     16.28    718.31    189.65
 MKDIR                0    0%        0
 NULLPROC            22    1%       64         22      3.48      7.09      8.87
 READ              1263   35%     2789       1262     11.97    466.80    329.74
 READDIR            109    3%      225        109     10.17    116.09     61.57
 READLINK            21    1%       47         21      8.80     76.02     25.08
 REMOVE               0    0%        0
 RENAME               1    0%        7          1     64.86               64.86
 RMDIR                0    0%        0
 SETATTR              0    0%        0
 STATFS              11    0%       24         11      2.47      0.87      4.01
 SYMLINK              0    0%        0
 WCACHE               0    0%        0
 WRITE              500   14%     1444        500     69.28   1116.28    316.14
```

Figure 2: Nfswatch data for a twenty minute interval on one subnet

readdir), create files (create), and others. Our strategy in creating an analysis tool was to aggregate those NFS operations which exhibited similar response times, as observed within our environment, into "read", "write", "create" and "lookup" groupings (Figure 3). We then established parameterized thresholds in each grouping for latencies we considered "excellent", "good", "poor" and "bad" (Figure 4). The groupings and their respective response time thresholds undergo continuous refinement as we analyze our data.

	Performance Criteria (in milliseconds)			
	Excellent	Good	Poor	Bad
LOOKUP	<4	<10	<30	>30
READ	<20	<40	<60	>60
CREATE	<35	<70	<100	>100
WRITE	<50	<100	<150	>150

Figure 4: Current thresholds for operation groupings

Our analysis tools have two distinct purposes. The first is to characterize all networks in a very general way for "continuous improvement" purposes. We want to be able to quantify the response time character of each subnet in a way that can be presented (perhaps too) simply to management, and use this "figure of merit" to benchmark, in a very general way, our month to month success at providing an ever better level of service. It is very important that such a figure of merit "improve" whenever there is a real improvement and "degrade" whenever service has deteriorated. We would like a kind of average response time, but we need to identify unacceptable response times whenever they occur. Providing thresholds for each operation grouping helps identify "bad" response times, and helps normalize the response times to allow averages across all operations without having variations in operation mix generate false signals of network quality variation.

Our second objective is to have the ability to diagnose server and/or filesystem level problems at the operation grouping level, apply appropriate remedies, and verify their success. We have have created two perl tools "rpcspy2fom.pl" and "rpcspy2art.pl" to support these objectives.

Figure 5 illustrates the results of our analysis. Filesystems which exhibit an aggregate delay in the "bad" operation category of more than five seconds are placed on a "bad filesystem list", ordered by the size of their aggregate delay. The behavior of "excellent", "good" and "poor" operations is summarized in a single "figure of merit" for the subnet. We characterize the service quality of a subnet both by the "figure of merit" and by the number of filesystems which show up on its list. To focus our attention on the worst problems, we sort the lists of bad filesystems globally across all networks for cumulative time spent in "bad" operations.

```
                    Average Response Time for NFS Operations

                        (Read)    (Write)    (Create)    (Lookup)    (n/a)
( 1) CREATE     =                              16.6
( 2) GETATTR    =                                           3.4
( 3) GETROOT    =                                                       0
( 4) LINK       =                              13.9
( 5) LOOKUP     =                                           7.0
( 6) MKDIR      =                102.9
( 7) NULL       =                                           2.8
( 8) READ       =  11.7
( 9) READDIR    =  11.5
(10) READLINK   =                                           4.5
(11) REMOVE     =                              30.9
(12) RENAME     =                              28.1
(13) RMDIR      =                              40.7
(14) SETATTR    =  11.2
(15) STATFS     =                                           3.4
(16) SYMLINK    =                              13.4
(17) WCACHE     =                                                       0
(18) WRITE      =                 60.2
```

Figure 3: Aggregate NFS response time data from 12 subnets (4 months)

Results and Conclusion

Our NFS response time analysis framework has enabled us to detect and remedy specific write, read and lookup category problems. For example, installing a write cache on an Auspex server showed a dramatic effect in its response time profile. In other cases we have seen the need to add memory to machines exhibiting slow lookup category response times and to devise new filesystem distributions for machines with slow read category response times. In each case, we are able to use historical data to perform a before-and-after analysis of our success at dealing with these problems. We are also able to present each month a management measure of network quality based upon NFS response time.

Availability

Nfswatch v4.0 and rpcspy can be obtained from Internet archive sites. Our modifications to rpcspy, and the perl programs used in our analysis can be obtained by sending an email request to gls@cirrus.com.

Author Information

Gary L. Schaps is an analyst in the computing resources department at Cirrus Logic. He has an MSCS degree from the University of Miami and enjoys writing software, managing networks and playing raquetball. Reach Gary at: gls@cirrus.com.

Peter Bishop holds a Ph.D. in computer science from MIT, and manages the computing resources department at Cirrus Logic. Reach Peter at: peter@cirrus.com.

References

R. Lehman, G. Carpenter, & H. Hien, "Concurrent Network Management with a Distributed Management Tool", *USENIX LISA VI Conference Proceedings, 1992.*

NFS RESPONSE TIME REPORT NET: 141.131.99 DATES: 8/1/93 - 8/13/93
FIGURE OF MERIT: 0.130 THRESHOLD: 5 seconds

Server Filesys Op	Excellent	Good	Poor	Bad
ss212 sd2c Write	3079 <50ms 40.5ms 0.6%	41072 <100ms 70.4ms 7.5%	157114<150ms 129.ms 28.7%	346748>150ms 184.ms 63.3%
ss212 sd3c Write	653 <50ms 44.4ms 0.3%	7551 <100ms 82.8ms 3.6%	76833 <150ms 134.ms 36.6%	125067>150ms 174.ms 59.5%
sungraf sd12d Write	7 <50ms 34.9ms 0.0%	7751 <100ms 89.6ms 7.6%	27601 <150ms 108.ms 27.1%	66482 >150ms 186.ms 65.3%
ss301 sd2c Write	71 <50ms 31.8ms 0.2%	1746 <100ms 74.5ms 5.5%	5593 <150ms 147.ms 17.6%	24447 >150ms 177.ms 76.7%
ss212 sd2c Lookup	86885 <10ms 2.7ms 93.5%	3179 <20ms 14.5ms 3.4%	1178 <50ms 23.0ms 1.3%	1688 >50ms 2147ms 1.8%
sungraf sd8a Write	40 <50ms 36.2ms 0.4%	2 <100ms 81.ms 0.0%	706 <150ms 119.ms 7.8%	8310 >150ms 266.ms 91.7%
tango vp30 Write	401503<50ms 31.3ms50.4%	307670<100ms 68.9ms 38.6%	80988 <150ms 112.ms 10.2%	6664 >150ms 151.ms 0.8%
talisman sd5g Write	19191 <50ms 33.8ms26.9%	19003 <100ms 86.8ms 26.6%	29137 <150ms 113.ms 40.8%	4027 >150ms 226.ms 5.6%
sunrise sd7g Write	2 <50ms 48.ms 0.1%	80 <100ms 71.1ms 2.4%	3 <150ms 116.ms 0.1%	3201 >150ms 233.ms 97.4%
tango ad0f Write	163 <50ms 47.9ms 4.2%	738 <100ms 61.4ms 19.0%	165 <150ms 145.ms 4.3%	2813 >150ms 264.ms 72.5%
talisman sd6c Create	9136 <40ms 29.4ms76.7%	2710 <70ms 46.4ms 22.8%	9 <100ms 86.3ms 0.1%	52 >100ms 142.ms 0.4%
talisman id000a Create	1429 <40ms 17.3ms36.3%	2102 <70ms 57.1ms 53.5%	348 <100ms 77.7ms 8.9%	53 >100ms 129.ms 1.3%
talisman id001g Lookup	6185 <10ms 1.9ms 75.7%	1754 <20ms 15.3ms 21.5%	182 <50ms 30.9ms 2.2%	48 >50ms 114.ms 0.6%

Figure 5: Annotated NFS Repsonse Time Report

Bruce E. Keith, "Perspectives on NFS File Server Performance Characterization", *USENIX Summer Conference Proceedings, 1990.*

Barry Shein, Mike Callahan & Paul Woodbury, "NFSSTONE A Network File Server Performance Benchmark", *USENIX Summer Conference Proceedings, 1989.*

Andy Watson & Bruce Nelson, "LADDIS: A Multi-Vendor and Vendor Neutral SPEC NFS Benchmark", *USENIX LISA VI Conference Proceedings, 1992.*

Matt Blaze, "NFS Tracing By Passive Network Monitoring", *USENIX Winter Conference Proceedings, 1992.*

David A. Curry and Jeffrey C. Mogul, nfsstat man page, 1993.

AIM Technology, *WireTap User Guide (Version 1.1.3)*, 1992.

The Amanda Network Backup Manager

James da Silva & Ólafur Guðmundsson
– Department of Computer Science, University of Maryland

ABSTRACT

We present *Amanda*, a freely redistributable network backup manager written at the University of Maryland. Amanda is designed to make backing up large networks of data-full workstations to gigabyte tape drives automatic and efficient.

Amanda runs on top of standard Unix backup tools such as `dump` and `tar`. It takes care of balancing the backup schedule and handling any problems that arise. Amanda runs backups in parallel to insure a reasonable run time for the nightly backups, even in the presence of slow computers on the network. Tape labeling insures that the wrong tape is not overwritten. A report detailing any problems is mailed to the system administrator in the morning.

In our department, we use Amanda to back up about 35 gigabytes of data in 336 filesystems on more than 130 workstations, using a single 5 gigabyte 8mm tape drive. Nightly runs typically complete in three to four hours. Amanda is currently in daily use at sites around the world.

Motivation

Until a few years ago, the backup medium of choice for most large Unix sites was the 9 track reel-to-reel tape, while 1/4" cartridge tapes were (and still are) popular with smaller systems. Storage capacities for 9-track and cartridge tapes vary from about 40 to 200 Megabytes. These tape systems are often of smaller capacity than the disk subsystems they are backing up, requiring an operator to feed multiple tapes into the drive for a full backup of the disks.

This problem has had a big influence on large site system administration. Sites with only a few large timesharing systems or file servers can arrange backups by operators at scheduled times, but the coordination of backups of a large number of workstations on a network is more difficult. Requiring users to do their own backups to cartridge tapes doesn't work very well; even computer-literate users just don't do backups on a regular basis.

A solution that many sites have adopted is a *dataless* workstation model, in which all user data is stored on file servers, with small local disks to hold temporary files and frequently used binaries, or even a *diskless* workstation model, where the workstations have no disks at all [1]. These network organizations require fast file servers with large disks, and generate heavy network traffic.

Our department, on the other hand, has always used *datafull* workstations, where all user data, temporary files, and some binaries, are stored on the workstations. File servers only provide shared binaries. This allows the use of smaller file servers, with smaller disks. A big advantage of this model is political; users tend to want their own disks with their own data on their own desks. They don't want to deal with a central authority for space or CPU cycles, or be at the whim of some file server in the basement.

Since most file writes are local, network traffic is lower and expensive synchronous NFS file writes are avoided, improving performance [2]. With the datafull model we are able to have each fileserver support over 40 machines if needed, while in dataless and diskless environments only specialized fileservers can support more than 20 workstations. The big disadvantage is the difficulty of managing and backing up all the datafull workstations.

The arrival of inexpensive gigabyte Digital Audio Tape (DAT) and 8mm video tape technology changed the situation drastically. Affordable disks are now *smaller* than affordable tape drives, allowing the backup of many disks onto a single gigabyte tape. It is now possible to back up all the workstation disks at a site over the network onto a single 8mm tape.

With the space problem solved, the new problem is *time*. Backing up workstations one at a time over the network to tape is simply *too slow*. We found that we could not add workstations to our network backups because the nightly backup would not finish until well after the start of the next work day. Many workstations cannot produce backup data as quickly as tapes can write [3]. For example, typical backup rates (both full and incremental) on our network range between about 5% to 70% of the rated 246 KB per second of our Exabyte EXB-8200 8mm tape drives [4].

Amanda, the "Advanced Maryland Automated Network Disk Archiver," was developed to solve these problems. To make the project manageable, we first built Amanda on top of the standard BSD Unix dump program. Amanda uses an optional *holding disk* to run multiple backups in parallel, and copies the backup images from the holding disk to tape, often as fast as the tape can stream. This version was described in [5].

More recently, we have be working on generalizing Amanda to handle backup programs other than BSD dump, like tar (and potentially PCs and Macintoshes in the future), and adding support for Kerberos-style authentication and data encryption. Meanwhile our site has grown from 10 gigabytes of data backed up with Amanda, to 35 gigabytes, and we have moved to a 5 gigabyte tape drive.

This paper concentrates on the features of Amanda as seen from the point of view of the system administrator and operators. We will touch on configuration possibilities, daily operation, restores, reported problems, backup data integrity, and have a look at the performance of Amanda at our site for the past year and a half. We conclude with a comparison of Amanda with some other free and commercial network backup systems.

Amanda Overview

Amanda is designed to back up a large network of computers (*hosts*) to a Unix host with a gigabyte or larger tape drive. The host with the tape drive, known as the *backup server host*, can optionally contain a *holding disk*, which is used as a staging area for parallel backups. While the holding disk is optional, a relatively large disk is recommended for high performance. Depending on the site, from 200 MB up to 1 GB of holding disk can be effectively used to speed up backups. Without the holding disk, backup rates are limited to the rate at which individual hosts can generate backup data sequentially.

Amanda backups are intended to be run in the middle of the night from cron on the backup server host. This server host communicates with Amanda programs running via inetd on all the hosts to be backed up, known as the *backup client hosts*. When all the night's backups are completed, a detailed mail report is sent to the system administrators.

The server host program is amdump, which consists of several distinct submodules that can report results to the user. planner is the backup cycle scheduler; it determines what level each filesystem will back up at each night. driver manages the nightly run and orchestrates the actual flow of backups. dumper communicates with each client host, and taper drives the tape device. On the client hosts, amandad is invoked (via inetd) by requests from the server host.

In addition to the main overnight backup program, Amanda has several auxiliary programs:

1. amadmin is the general purpose administrator's utility. Amadmin encapsulates a number of small functions, like database and log queries.
2. amrestore restores backups from Amanda tapes. It takes care of finding the right filesystem's backup on the tape and piping the backup data to the underlying restore program.
3. amcheck is usually run in the afternoon to make sure that everything is set up correctly for the next amdump run. It sends mail reporting any potential problems to the system administrators so that the problems can be fixed before the night's run. In particular, amcheck makes sure the correct tape is loaded into the tape drive, and checks for common problems on the server and all the client hosts, such as permissions problems or nonexistent filesystems.
4. amflush writes backup files from the holding disk onto tape. If amdump detects a tape error, it will still try to back up as much data as possible to a holding disk on the server host, to avoid complete failure of the nightly backups. amflush is run by an operator the next day after the tape problem is corrected.
5. amlabel writes Amanda labels onto fresh tapes.
6. amcleanup recovers after any crash in the middle of an amdump run. It is usually run at boot time, and takes care of sending the mail report so that the system administrators know that backups were interrupted.

Configuration

Amanda is organized around *configurations*. Each configuration backs up a list of filesystems to a particular tape drive using a particular schedule. Multiple configurations can co-exist on a single server host. This can be useful for separating archives from daily backups, or balancing filesystems between tape drives.

Configuration Files

The Amanda programs are driven completely by two simple files maintained by the system administrators. The configuration file, amanda.conf, gives settings for a number of parameters. The disklist file contains a one-line entry for each filesystem to be backed up.

An example amanda.conf file is shown in Figure 1. This file is the central control panel for all Amanda activity. A number of parameters can be controlled by the system administrator to customize the backups to taste. Some of the possibilities are discussed in more detail below.

The disklist file merely lists all the filesystems that are to be backed up by this Amanda configuration, like so:

```
# hostname diskdev dumptype
salty      sd0a    comp-root
salty      sd0g    comp-user
```

The host name and device name for the partition are given, followed by the *dump type* name. This name refers back to an `amanda.conf` definition which specifies various per-filesystem parameters.

The Backup Schedule

Amanda manages the backup schedule within the parameters set in `amanda.conf`. It will move up full backups to balance the size of each night's run across the whole schedule, but will never delay a full backup for balancing purposes.

The configuration files allow many styles of backup schedule to be implemented with Amanda. Some of these are:

- **Periodic Full Backups with Daily Incrementals:** This is the most common style of backup. The backup schedule is set to some number of weeks (i.e. set `mincycle 2 weeks` in `amanda.conf`). Each filesystem will normally get a full backup once within this cycle, and an incremental backup every other night. The full backups can be moved forward at Amanda's discretion to balance the schedule.
- **Periodic Archival Backups:** An Amanda configuration can be set up that does just full backups to a new tape each time. These tapes are then archived permanently. Set

  ```
  options skip-incr, no-compress
  ```

```
org "CSD"                 # your organization name for reports
mailto "csd-amanda"       # the mailing list for operators at your site
dumpuser "bin"            # the user to run dumps under

inparallel 8              # maximum dumpers that will run in parallel
netusage  500             # maximum net bandwidth for Amanda, in KB per sec

mincycle  10 days         # the number of days in the normal dump cycle
tapecycle 20 days         # the number of tapes in rotation
bumpsize 10 MB            # minimum savings (threshold) to bump level 1 -> 2
bumpdays     2            # minimum days at each level
bumpmult     2            # threshold = bumpsize * (level-1)**bumpmult

tapedev "/dev/nrst8"      # the tape device
tapetype EXB-8500         # what kind of tape it is (see tapetypes below)
labelstr "^VOL[0-9][0-9]*$"     # label constraint regex: all tapes must match

diskdir "/amanda2/amanda/work"  # where the holding disk is
disksize 800 MB                 # how much space can we use on it

infofile "/usr/adm/amanda/csd/curinfo"  # database filename
logfile  "/usr/adm/amanda/csd/log"      # log filename

define tapetype EXB-8500 {     # specifies parameters of our tape drive
    length 4200 mbytes
    filemark 48 kbytes
    speed 480 kbytes
}

define dumptype comp-user {    # specifies parameters for backups
    program "DUMP"
    options compress           # compression is optional
    priority medium
}

define dumptype comp-root {
    program "DUMP"             # DUMP or GNUTAR or ...
    options compress
    priority low        # root partitions can be left for last
}
```

Figure 1: Example Configuration

in the dump type specifications to turn off incrementals and compression, and set

```
tapecycle inf
```

to tell Amanda that the tapes are never cycled.

- **Incremental Only, with external full backups**: Large timesharing hosts that are always active are best backed up by hand in single user mode during a scheduled down-time period. The daily backups can still be done with Amanda, by specifying `options skip-full` on those filesystems, and running `amadmin force` to lock the full backup position to the night the external backup is done. Thereafter Amanda will attempt to keep in sync with the external backup, and even warn the operators when the scheduled backup is due.
- **Incremental Only, with no full backups**: Some filesystems don't normally change at all relative to some reference filesystem. For example, root partitions are often derived from a site-wide standard prototype, plus small local customizations. These partitions can be installed such that incremental backups capture just the local changes. With `options no-full` in the dump type, Amanda will do incremental backups for these filesystems on each run, with no bumping (see below for a description of *bumping*).
- **Frequent Full Backups, No incrementals**: Some sites don't like to bother with incremental backups at all, instead doing full saves of all their disks each night, or as often as possible. Such a site can be run similarly to an archive configuration, with `options skip-incr` set for each disk, and `mincycle` set as low as possible given the size of the disks and the backup tape.

Automatic Incremental Bumping

Berkeley `dump` supports the concept of multiple *levels* of incremental backups, whereby a backup at level *n* backs up every file modfied since the last backup at level *n-1*. Other backup programs, such as `tar`, can be run in the same way.

The different backup levels allow a tradeoff between redundancy of data on tape, and saving tape space by only backing up recently changed files. Coming up with the right tradeoff can be a chore: experienced administrators will remember the ''Modified Tower of Hanoi algorithm'' recommended in the original Berkeley `dump` man pages.

Amanda is smart enough to only change the incremental level (known as *bumping*) for a filesystem when significant tape space would be saved by doing so. Amanda also takes care to not bump too eagerly, since having too many incremental levels makes full restores painful. Three `amanda.conf` parameters are provided for the system administrator to control how bumping is done.

- **`bumpsize`** Default: `10 MB`. The minimum savings required to trigger an automatic bump from incremental level one to level two. If Amanda determines that a level two backup will be this much less than a level one, it will do a level two.
- **`bumpmult`** Default: `2.0`. The bump multiplier. Amanda multiplies the bumpsize by this factor for each level. This prevents active filesystems from bumping too eagerly by making it harder to bump to the next level. For example, with the default bumpsize and bumpmult, the bump threshold will be 10 MB for level one, 20 MB for level two, 40 MB for level three, and so on: 80 MB, 160 MB, 320 MB, 640 MB, and finally 1280 MB savings required to bump from level eight to level nine.
- **`bumpdays`** Default: `2`. To insure redundancy in the backups, Amanda will keep filesystems at the same incremental level for at least bumpdays days, even if the bump threshold criteria are met.

Tape Management

Amanda supports the labeling of tapes to avoid overwriting active data or non-amanda tapes.

The `amlabel` command puts an Amanda label onto a fresh tape. The `tapecycle` parameter controls how many tapes are considered to be in active rotation. Normally there would be at least several more tapes in rotation than there are days in the backup cycle. This allows some slack should a machine be out of commision for several days.

Amanda labels are arbitrary names; the system administrator chooses the tape naming system. The `labelstr` configuration parameter constrains valid tape labels to a certain regular expression pattern. For example,

```
labelstr "^VOL[0-9][0-9]*$"
```

only allows labels of consisting of the prefix `VOL` followed by a number.

The `labelstr` facility can prevent two configurations using the same tape drive from overwriting each other's tapes. If each configuration uses a different label prefix, tapes from other configurations will be protected.

Daily Operation

Once Amanda is installed and configured, very little effort is required for daily operation. Adding and deleting filesystems from the backup list is as simple as editing the `disklist` file.

In addition to maintaining the `disklist`, the operators must change the tapes, handle any restore requests, read the nightly report generated after the backups complete, and deal with any problems mentioned in the reports.

Day-time Check

Since the Amanda backups are done in the middle of the night, presumably when no operators are around, it is important that possible failure modes are checked for before the run, when operators are present.

The `amcheck` program checks that the right tape is in the drive, and that there is enough room on the holding disk for proper operation. If not, it will send mail to the operators listing its complaints. `amcheck` is run from `cron` after the time the tape is normally changed, but early enough that someone can solve the problems before the run.

Figure 2 shows a sample of the amcheck mail generated when two problems occurred: the holding disk had less free space than requested in `amanda.conf`, and the wrong tape is in the tape drive. Both problems are most likely the result of an operator doing a restore from tape `VOL18` earlier in the day using the holding disk during the restore. The mail message reminds the operators to clean up after they are finished.

Reported Problems

After the nightly amdump run completes, mail is sent to the operators giving the details of the night's operations. Any errors are summarized at the very top of the report, with details given below. The report includes summary statistics as well as a line for each filesystem, telling of its success or failure and how it performed.

An excerpt of a nightly report is given in Figure 3. In this example, one of hosts (**`idaho`**) is down, and a filesystem on **`rath`** has developed a bad spot. Even though dump continues after read errors and eventually succeeds, Amanda catches the problem by scanning through the dump message output for anything interesting. If unknown patterns pop up, the dump output is displayed for the operators to deal with the problem. In this case, the filesystem in question should be reformatted and restored.

Amanda catches a number of common problems, including:

- As in the example, *disk errors* that occur during backup are brought to the operators' attention. This allows them to be detected and corrected very quickly.
- Any other *backup program errors*, such as permission problems, or even a core dump, are caught and brought to the operators' attention.
- Any *down client hosts* are identified by Amanda. Their filesystems are failed, giving them a higher priority the next run.
- Any *backups that hang* are detected; Amanda times out if no backup data is received for a certain time.
- If the *wrong tape* is in the tape drive, Amanda will not overwrite it. Instead it writes, in priority order, as many incremental backups to the holding disk as will fit. These can be put onto the next tape with the `amflush` command.

In addition to identifying problems, the report gives many vital statistics and *notes* from the various subsystems. In Figure 3 we see several notes from `planner`. Any bumps of incremental levels or promotions of full backups from later in the schedule are mentioned. In addition, we see that the operators have requested that a filesystem be forced to a full backup on this run. `planner` confirms in the report that the full backup will be done.

Restores

There are two phases to doing a restore. First, the correct tapes to restore from must be determined, and second, the data must be retrieved from the tape.

The `amadmin find` command shows the backup history for a particular filesystem. Consider the following example output:

```
date      host disk lv tape  file stat
93-09-11  rath sd0g 1  VOL2   323 OK
93-09-10  rath sd0g 1  VOL1   305 OK
93-09-09  rath sd0g 1  VOL20  262 OK
93-09-08  rath sd0g 1  VOL19  242 OK
93-09-07  rath sd0g 1  VOL18  127 OK
93-09-04  rath sd0g 0  VOL17   99 OK
```

To do a full restore of this filesystem, only tapes `VOL17` and `VOL2` need to be restored. To restore a single user file or directory, more information is needed. For example, a user might create a file on

```
From:    bin
To:      csd-amanda
Subject: CSD AMANDA PROBLEM: FIX BEFORE RUN, IF POSSIBLE

WARNING: disk space low: 552972 KB avail < 884736 KB requested.
         (please clear out cruft from /amanda2/amanda/work's partition)
ERROR: cannot overwrite active tape VOL18.
       (expecting tape VOL2 or a new tape)
```

Figure 2: Example amcheck report

```
From:    bin
To:      csd-amanda
Subject: CSD AMANDA MAIL REPORT FOR September 11, 1993

These dumps were to tape VOL2.
Tonight's dumps should go onto tape VOL3 or a new tape.

FAILURE AND STRANGE DUMP SUMMARY:
  idaho       sd2h lev 0 FAILED [could not connect to idaho]
  rath        sd0a lev 1 STRANGE

STATISTICS:                 Total       Full      Daily
                          --------   --------   --------
Dump Time (hrs:min)          3:38       1:57       1:17   (0:12 start, 0:12 idle)
Output Size (meg)          2709.8     1796.3      913.5
Original Size (meg)        4881.7     3044.0     1837.7
Avg Compressed Size (%)      51.4       53.4       48.5
Tape Used (%)                64.9       42.8       22.1   (level:#disks ...)
Filesystems Dumped            335         26        309   (1:276 2:26 3:5 4:2)
Avg Dump Rate (k/s)          48.8       56.6       38.4
Avg Tp Write Rate (k/s)     238.1      262.1      201.8

FAILED AND STRANGE DUMP DETAILS:

/-- rath        sd0a lev 1 STRANGE
| senddump: start rath sd0a level 1 to amanda.cs.umd.edu
|   DUMP: Date of this level 1 dump: Thu Sep  9 01:38:51 1993
|   DUMP: Date of last level 0 dump: Thu Sep  2 01:58:25 1993
|   DUMP: Dumping /dev/rsd0a (/) to standard output
|   DUMP: mapping (Pass I) [regular files]
|   DUMP: mapping (Pass II) [directories]
|   DUMP: estimated 786 blocks (393KB) on 0.00 tape(s).
|   DUMP: dumping (Pass III) [directories]
|   DUMP: dumping (Pass IV) [regular files]
?   DUMP: (This should not happen)bread from /dev/rsd0a [block 6992]: ...
|   DUMP: level 1 dump on Thu Sep  9 01:38:51 1993
|   DUMP: 790 blocks (395KB) on 1 volume
|   DUMP: DUMP IS DONE
| senddump: end
\--------

NOTES:
  planner: Forcing full dump of tove:sd0a as directed.
  planner: Incremental of cortex:sd0g bumped to level 3.
  planner: Full dump of lovedog:rz9g promoted from 1 days ahead.

DUMP SUMMARY:
                                         DUMPER STATS                        TAPER STATS
  HOSTNAME     DISK    LV  ORIG-KB    OUT-KB  COMP%   MMM:SS    KB/s   MMM:SS    KB/s
  --------------------- ------------------------------------------------  --------------
  idaho        sd0a     1  FAILED -------------------------------------------------------
  idaho        sd0h     1  FAILED -------------------------------------------------------
  idaho        sd2h     0  FAILED -------------------------------------------------------
  lovedog      rz3a     1      403      128   31.8     0:04    35.6     0:03    57.8
  lovedog      rz3g     3     9745     1678   17.2     1:14    22.5     0:09   192.4
```

Figure 3: Excerpt from Nightly Amanda Report

September 7 then accidentally delete it on 9th, and want it back a few days later. In this case VOL19 must be restored to get the file. The restores are done with the `amrestore` program. `amrestore` gets the proper backup off of the Amanda tape and outputs the backup image. This can be put on a staging disk (the holding disk works well for this), or piped directly to the restore program.

For example, to do a full restore of `rath's` `sd0g` disk from `rath`, the command would be:

```
rsh amanda amrestore -p /dev/nrst8 \
        rath sd0g | restore xf -
```

where `amanda` is the Amanda tape server host.

Data Integrity

There are two major issues affecting the integrity of backup data that system administrators need to keep in mind when designing their backup system. First is the online backup problem, the second is compression.

Online Backups

The Online backup problem is well-known and has been discussed in previous LISA papers [6, 7]. As Shumway shows, it is impossible in general to insure completely correct backups on an active filesystem without operating system support. Adding, modifying, deleting, and moving files and directory trees while the backup is running can cause data to be missed, or worse, confuse the backup program into crashing or generating a corrupted output that cannot be restored.

Amanda suffers from this problem to the same extent that the underlying backup program does. If the vendor's backup program does not make system calls to lock out filesystem changes at sensitive times, then the potential for problems exists. Unfortunately, most vendors' operating systems do not have such a facility.

In practice, it turns out that the effect of this problem is small. For most filesystems on user workstations, very little is going on in the middle of the night. Since the technology to solve the problem is not yet generally available, an administrator faced with backing up dozens or hundreds of filesystems has little choice but to take the risk and do online backups.

For very active filesystems, like those on large timesharing systems or 24 hour database engines, it is probably still best to do full backups the old fashioned way, by bringing the machine down to single user mode for regularly scheduled backups. On such a system, Amanda can still be used for daily incremental backups.

Compression

Compression is completely optional in Amanda; it can be turned on or off on a per-filesystem basis.

Compression has a negative effect on the ability to restore from partially damaged backup images. The standard Unix uncompression program will be confused by the first error, causing the rest of the backup image to be lost or garbled.

For this reason, compression of data on long-term, archival backups is not recommended, as the chance of tape errors increases with long term storage. However, for tapes in a short term backup rotation, the chances of errors is small if proper care is taken of the tapes and the drive. In this situation, compression of backups is not much risk, and is worth the benefit of more than doubling the amount of data that will fit on each tape.

Turning off compression is no guarantee that errors can be recovered from. Some vendors' tape drivers will not keep reading after a medium error. A system administrator that is counting on this to work should test the hardware and software carefully. A strong magnet applied to a loop of tape somewhere in the middle of a large backup file can produce surprising results.

Backups at `CS.UMD.EDU`

Amanda's home site is the Computer Science Department of the University of Maryland at College Park. Here we have been running the parallel version of Amanda for over a year and a half, keeping statistics the entire time.

Figure 4 shows the growth in the data on the hosts being backed up by Amanda at our site. This does not include two active timesharing systems, and some of the active file server disks, which are still backed up by hand in single user mode (these non-Amanda disks add about another 8 GB to the site size).

After an initial test period from January to March, 1992, we brought all the workstations in the department onto the Amanda backups by the summer of 1992. All the growth since that time has been from bringing more data online. The plunging cost of gigabyte disk drives has had a dramatic affect on the department; the amount of data on CSD disks more than doubled, from about 15 GB in September 1992, to over 35 GB in September 1993.

We expect that other departmental level sites are seeing similar growth rates. Given the current availability of inexpensive 2 GB drives and user's insatiable demands for disk space, it seems reasonable to expect continued large increases in the amount of data system administrators are expected to back up.

Luckily, the amount of data that needs to be written to tape each night grows much more slowly. Use of compression divides the growth rate in half, and a two week backup cycle divides it again by ten. When the nightly backup reaches capacity, the backup cycle can be extended. Amanda's automatic bumping relieves the increased pressure of incremental backups in this situation.

In CSD our original 2 GB EXB-8200 became uncomfortably full in September 1992. We extended our backup cycle to three weeks, which kept us going until we brought the 5 GB EXB-8500 on-line in January 1993.

Amanda has also done a good job of holding down the backup times in the face of fast growth, as can be seen from Figure 5, which shows each of the nightly amdump run times. The run time has stayed for the most part in the 3 to 4 hour range. Interestingly, the variance in run times has increased considerably, with the occasional run taking more than 6 hours.

The number of short or completely failed runs have reduced, as the operators have gotten into the routine. One run in particular stands out: In August 1992 an operator added a 300 MB filesystem on a very, very slow Sun 2 with compression turned on. That disk alone took almost twelve hours to complete a full backup. Needless to say, we turned off compression for that disk the next night!

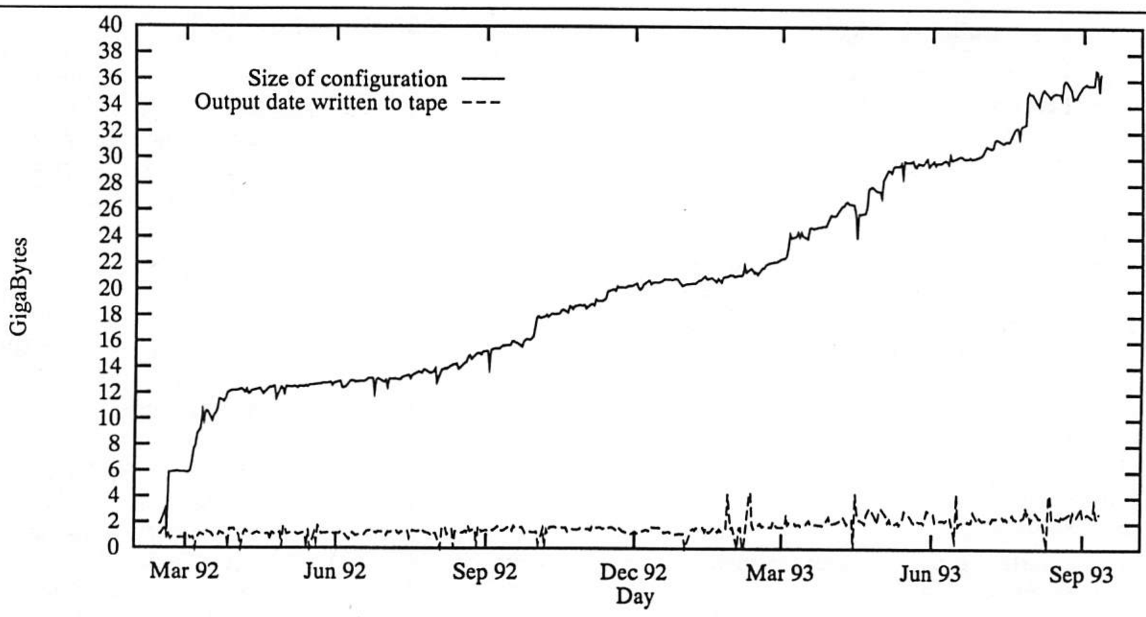

Figure 4: Nightly Amanda Backup Size at `CS.UMD.EDU`

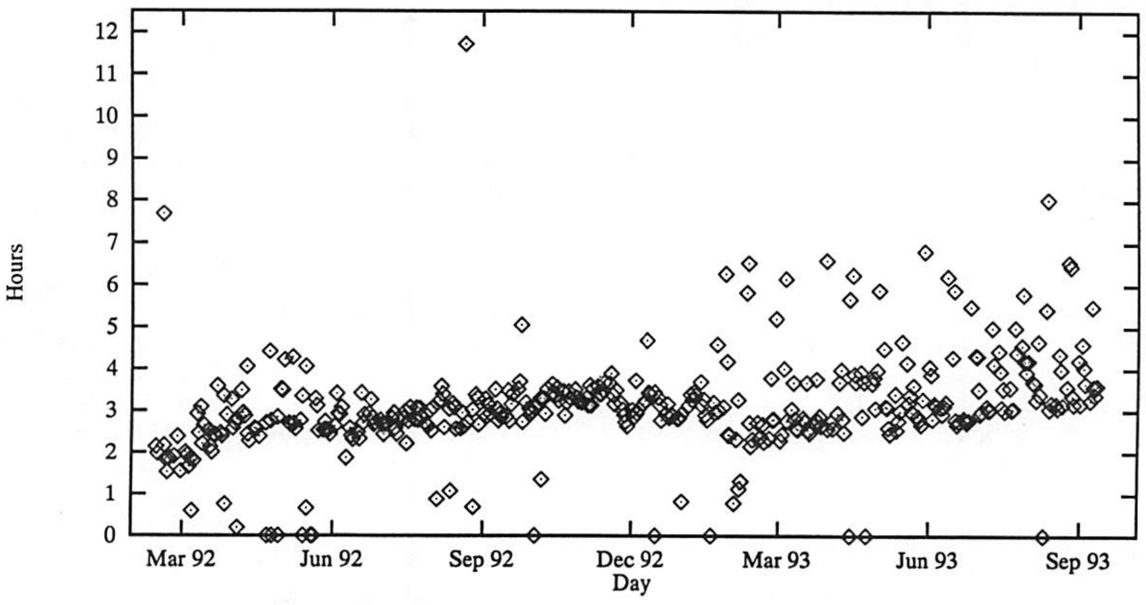

Figure 5: Nightly Amanda Run Time at `CS.UMD.EDU`

Comparisons with other Backup Systems

There are a number of systems available that perform similar functions as Amanda. This section makes no judgement, but will highlight some of the similarities and the major differences. The systems that we examined for this study that are freely distributed on the Internet are:

- Amanda from University of Maryland [5]
- Backup-2.6 from Ohio State University [1, 8]
- CUCCS Network Backup System from Carleton University (CUCCSNB) [9]
- DeeJay from Columbia University [10]

We also looked at three of the commercially available products:

- Budtool from Delta Microsystems [11]
- EpochBackup from Epoch Systems [12]
- Networker from Legato Systems [13]

All the systems above are designed to perform the same function, that is: *back up a heterogenous network of computers to large tapes, without an operator present.* The main differences are in the approach taken by the different tools. There are many different ideas about the "Right Way" to perform backups, and the tools reviewed have chosen different policies.

This is not a complete list of available systems but it is a good cross section. Some systems we did not look at are vendor specific and thus useless in a heterogenous network.

Approaches to Parallelism

One of the most common approaches to performing the backups in limited time is to divide the site into multiple partitions, with each one going to its own tape drive, and perform the backups in each partition sequentially. Once the partitions are in place the system should be rather stable, but some support is required to balance the load across the partitions, and to select the appropriate partition for additions. Load balancing may have to be done for both space and time.

A further advantage of this approach is that it is simple, and single tape failures affect only some of the hosts. The main disadvantage is low tape utilization due to low backup rates from hosts. Another disadvantage is that when configurations are highly loaded, operators may have to reorganize and load balance frequently.

Staging the backups to a disk is a slightly more complex approach, but it is less expensive than the one above, as it can utilize the tape better. In this scheme backups are performed at their natural speed to a holding disk, and then transferred to tape at high speed. This allows more backups to fit in each configuration. It is more reliable, as the staging disk can be used to store emergency incremental backups when there is a tape problem.

Writing multiple parallel backups to tape is the most complex approach, as this requires a special tape format. Of the systems we looked at, only Legato Networker uses this approach. This approach should outperform the other two in backup speed, but at the cost of complexity, non standard tape format, and slower restores (as the data for a particular disk will be spread out on the tape).

Backup Scheduling

The simplest way of performing backups is to always backup filesystems in the same order. In this scheme the variable is the level each filesystem is backed up at. Systems like Backup-2.6, Networker, CNCCS Network Backup and DeeJay use this method exclusively. Epoch and Budtool support this mode along with other modes. The problem with this scheduling is that tape utilization must be kept low to accommodate differences in backup sizes between nights.

A slightly more intelligent scheduling takes into account the size of the backups and moves full backups around to balance the nightly backup size.

Another approach is to perform only incremental backups using the automated system during the week and then have operators perform the full backups over the weekend. Epoch, Budtool and Amanda allow the user to specify exactly on what days full backups will be performed.

Some systems allow the system administrator to force a full backup of a set of hosts on selected days. Other options are to skip certain days.

Intelligent scheduling allows systems to fit more disks on each tape and to perform backups in less time. It is hard to evaluate from the literature available how well each system performs. In general, advanced scheduling requires less work of system administrators as the system performs the load balancing on the fly.

User Interfaces

One of the more striking differences between the systems examined is the sophistication of the user interfaces. The commercial systems all have what seem to be nice graphical front ends, some for the system administrators and others that the end users can use to request restores. None of the free systems have any graphical front ends, but some have programs to generate graphical performance information.

The command interfaces for the free systems vary from rudimentary to full description languages. Without playing with the interfaces it is difficult to assess which ones are appropriately matched to the system features.

All the systems offer some reporting, ranging from reporting only errors to full status reports. It is hard to compare the systems as most do not document what exactly is reported and in what form. It seems that the commercial systems have superior reporting facilities. The important thing to look for is whether the reports include enough information, highlight all discrepancies, and give some hints to novice operators what the problem may be.

Backup Programs

In table 1 we list the underlying backup programs each system supports:

Error recovery

There are number of things that can go wrong each time a backup is to be performed. One of the most common errors is that the right tape is not in the tape drive. Jukeboxes are less likely to suffer from this problem. All the systems have some mechanism to check if there is a tape in the drive and it is the right one. The systems that support carousels have an advantage, as they can automatically change the tape to the correct one.

In a large installation it is not uncommon that some hosts fail each night for various reasons. Most systems handle this to some extent, but the static schedule systems may have some difficulty overcoming this problem as this can delay the next night's backup significantly, or cause full backups to be skipped.

Restores

The reason people do backups is of course to be able to perform restores. The speed of restores is important to many. It is limited by a number of factors: where the data is on the tape, how fast it can be accessed, and how many tapes need to be scanned to search for the data. All the commercial systems have full file catalogues that allow them to identify quickly which tapes to restore from. DeeJay and CUCCS Network Backup support this feature, Backup-2.6 and Amanda both plan to support this in the future.

Epoch and Budtool offer graphical tools that end users can use to select files to be restored, and the requests can even be handled without operator assistance, if the tapes are available in a carousel. All others seem to require the operator to do most of the work when restoring, and use textual tools for this operation.

On the other hand, when full restores of a disk have to be done it seems that most of the systems will take similar time, depending on how incremental backups are performed and how many levels of backups have been done. All the systems seem to allow restores to remote hosts.

Per-System Highlights

In *Amanda* all scheduling and configuration is done on the tape server host. This means that no new files are created on the other machines: only `.rhosts` and `inetd.conf` have to be changed. Amanda is invoked the same way each time. Generally, all the system administrators need to do once the system is operational is to add or delete disks. Load balancing is performed by the system. Operator intervention is required for restores and after tape failures to `amflush` data from the holding disk to tape.

Ohio State University *Backup-2.6* has the ability to backup each host multiple times each night to different tapes to prevent data loss from bad tapes. It also has an explicit support for off site storage of tapes. Great care has been put into this system to allow it to overcome all kinds of problems with data loss and site errors, but it has not been tuned as much for performance as some of the other ones. Due to its inflexible scheduling, system administrators must perform operations to load balance the system including delaying adding new disks.

Carleton University *Network Backup* is designed more from the mainframe point of view. It supports index files, tar and dump, and knows about administrative domains. The system is designed to allow a central facility to backup many administrative domains. It and its tools are only supposed to be used by a hierarchy of system administrators, and there are controls on what each level can do. It has multiple configurations and supports PCs to some extent, but at the same time it is not geared at the large populations that Amanda and OSUB handle so well.

DeeJay was designed around a carousel and incorporates advanced tape management for backup of many machines. The system manages the tapes as one infinite tape. Because the carousel has multiple tape drives, it can perform backups to each one at

	Dump	GNU TAR	CPIO	Special	Index
Amanda-2.2	x	x	x		
Backup-2.6	x	x			
DeeJay	x		x		x
CUCCSNB	x	x	x		x
Budtool	x	x	x		
EpochBackup				x	x
Networker				x	x

Table 1: Comparison of Backup Programs

the same time. DeeJay has a fixed schedule of full and incremental backups for each disk: the options are weekly, monthly, or never.

Delta Microsystem's Budtool performs backups in parallel by controlling multiple tape drives on multiple hosts at the same time. It provides a simple setup procedure where users can specify the exact commands to be executed on each host to backup the system. It supports tar, dump and cpio, among others.

EpochBackup is in many aspects similar to Amanda: it provides a total hands off operation when use with EpochMigration. Unlike Amanda, EpochBackup does not run backups in parallel. Epoch claims that their special backup program is much faster than dump or tar. This system will detect changes in the configuration and notify system administrators if new disks are not being backed up. One of the advanced features claimed by this product is that restored directories will not contain deleted files, as tar based backup schemes will.

Legato *Networker*'s main distinction is that it uses nonstandard backup programs and tape formats. It performs parallel backups by multiplexing to the tape. This mechanism allows it to eliminate the holding disk, but at the cost of complex data format on the tape. Legato supplies clients for many Unix variants as well as for PC-DOS.

Future Directions

Amanda is still under active development. Some improvements not described in this paper are running in the lab (with varying degrees of solidity) and should be available about the time you read this, including:

- generalized backup program support, including `tar`, `cpio`, and eventually VMS, Macintosh, and PC-DOS clients.
- Kerberos Authentication, including sending encrypted data over the network.
- Generic carousel/stacker support. Supporting subsystems for particular hardware will need to be written.

In the longer term we are investigating the addition of a browseable file index, automatic tape verification, an X-based graphical user interface, writing to two tape drives at once, and interleaving backups on tape to allow good performance without a holding disk.

Availability

Amanda is copyrighted by the University of Maryland, but is freely distributable under terms similar to those of the MIT X11 or Berkeley BSD copyrights. The sources are available for anonymous ftp from ftp.cs.umd.edu in the pub/amanda directory.

There is also an active Internet mailing list for the discussion of Amanda, send mail to amanda-users-request@cs.umd.edu to join the list.

Author Information

James da Silva was a high-school hacker in the early '80s. He received a National Merit Scholarship from Georgia Tech in 1983, but soon left school to work in the Real World. He was a Systems Engineer for Electronic Data Systems, writing applications programs first on the PC, then under Unix. In 1987 he escaped the Real World and headed for the ivory towers of the University of Maryland. He finally got his degree after avoiding it for years by working on the Computer Science Department's computers rather than their classes. He now works there full time as a Faculty Research Assistant for the Systems Design and Analysis Group. Jaime can be reached at jds@cs.umd.edu.

Ólafur Guðmundsson was born in Reykjavík, Iceland. He graduated from the University of Iceland in 1983 with a B.S. in Computer Science. He worked as a systems programmer on VMS machines at the University of Iceland from 1983 to 1984. In 1984 he joined the graduate program of the Department of Computer Science at the University of Maryland where he had to learn some new operating systems, most of them unpleasant mainframe systems, until discovering Unix. Olafur obtained a Masters degree in 1987. Since then he has worked as a Faculty Research Assistant in the Department of Computer Science at University of Maryland. In this position he has been primarily involved in research, development and implementation of a distributed, hard-real-time operating system *Maruti*, but he has also worked on practical problems in computer networks and operating systems. Ólafur can be reached at ogud@cs.umd.edu or ogud@rhi.hi.is.

References

1. Steve M. Romig, "Backup at Ohio State, Take 2," *Proceedings of the Fourth Large Installation Systems Administration Conference*, pp. 137-141, The Usenix Association, Oct 1990.
2. Paul Anderson, "Effective Use of Local Workstation Disks in an NFS Network," *Proceedings of the Sixth Large Installation Systems Administration Conference*, pp. 1-7, The Usenix Association, Oct 1992.
3. Rob Kolstad, "A Next Step in Backup and Restore Technology," *Proceedings of the Fifth Large Installation Systems Administration Conference*, pp. 73-79, The Usenix Association, Sep 1991.
4. *EXB-8200 8mm Cartridge Tape Subsystem Product Specification,* Exabyte Corporation, Jan 1990.
5. James da Silva and Ólafur Guðmundsson, "Performance of a Parallel Network Backup

Manager,'' *Proceedings of the Summer 1992 Technical Conference*, pp. 217-225, The Usenix Association, Jun 1992.

6. Elizabeth D. Zwicky, ''Torture-testing Backup and Archive Programs: Things You Ought to Know But Probably Would Rather Not,'' *Proceedings of the Fifth Large Installation Systems Administration Conference*, pp. 181-190, The Usenix Association, Sep 1991.
7. Steve Shumway, ''Issues in On-line Backup,'' *Proceedings of the Fifth Large Installation Systems Administration Conference*, pp. 81-87, The Usenix Association, Sep 1991.
8. Steve Romig, ''The OSU-CIS Backup and Restore System,'' *Manual available from* `archive.cis.ohio-state.edu`, Ohio State University, Jan 1993.
9. R. Mallet, ''Carleton University Computing and Communications Services Network Backup Services,'' *Manual available from* `alfred.ccs.carleton.ca`, Carleton University, Apr 1992.
10. Melissa Metz and Howie Kaye, ''The Dump Jockey: A Heterogeneous Network Backup System,'' *Proceedings of the Sixth Large Installation Systems Administration Conference*, pp. 115-125, The Usenix Association, Oct 1992.
11. *BudTool Sales Literature,* Delta Microsystems, Inc., 1993.
12. *EpochBackup Technical Summary,* Epoch Systems, Inc., Oct 1992.
13. *Legato Networker Sales Literature,* Legato Systems, Inc., 1993.

PLOD: Keep Track of What You're Doing

Hal Pomeranz – QMS, Inc.

ABSTRACT

PLOD (the Personal LOgging Device) is a simple text interface which allows System Administrators (and others) to keep a record of the work they do from day to day. The program was developed in Perl with device independence, flexibility, extensibility, and ease of use in mind. The user-interface is reminiscent of Berkeley mail, complete with many pre-defined tilde-escapes which perform various useful functions. Users may easily extend the program by defining their own personal escape sequences.

Introduction

Certainly it may be said of System Administration that those who forget history are doomed to repeat it. However, running from crisis to crisis, as many Administrators do, is not conducive to a good memory: in fact, it is unlikely that anybody reading this can remember with detail what they were working on even one week ago. The solution, then, is to let some external device be your memory and then provide an easy means to recall information from this external store. PLOD, the Personal LOgging Device, is one such solution.

PLOD is really just a simple Perl program which will read lines from the keyboard and store these lines, along with a date/time stamp, into a (encrypted, if desired) log file. By default, a new log file is started automatically at the beginning of every month, although the exact time period may be customized by the user. A number of additional commands are supported, either on the command line or via tilde-escapes ala Berkeley mail.

The design goals for PLOD included device independence, flexibility, extensibility, and ease of use. Everything was done so that PLOD would be a tool that would get used frequently (several times per day). If not, there would be little point in PLOD's existence. Implementing the program in Perl went a long way towards achieving device independence, since Perl runs on everything from Amigas to VMS machines. The interface was specifically chosen to be one that required no assumptions about display device (i.e., simple text only). Much of PLOD's behavior can be customized via environment variables or initialization files. Users may even add their own escape sequences.

The Theory of Logging

Many and varied are the arguments about personal logs. Some think that logs are a complete waste of time, while others find them invaluable. Some see logs purely as a defense against management scrutiny, some see them as a storehouse for accumulated wisdom and trickery. Some keep personal information in their logs along with details of their professional work, others maintain a more rigid dividing line. Some people do a daily brain dump into their logs before they go home, others prefer to update their logs several times during the day. Should the log be hard copy or on-line? Ultimately, the decision is a personal one and must fit the way you do business.

In many organizations, the job of System Administration is not well understood by those not in the System Administration group. Some System Administrators spend inordinate amounts of time justifying their existence to management. In a confrontation, it can be difficult or impossible to regenerate the countless tasks that go into keeping enterprise systems functioning smoothly. Having documentation, in the form of a log, created prior to such a confrontation can be an invaluable aid.

Hopefully, things will never get to the confrontation stage: logs can help here, too. If you are not required to publish a regular status report, publish one anyway. Extract items from your logs which are of interest to your user community and publish them in a condensed format (nobody wants to read more than one or two screenfuls of information that don't pertain directly to their own work). It often helps to place notices of system downtime in these reports, so users will pay more attention to them. Make your management and your users understand what you do. If "they only notice you when something goes wrong", then there's an important sales job that you as a System Administrator are not doing right.

Take your managers to lunch every month and bring your logs along. Show them items that have contributed to your products getting out on schedule. Demonstrate areas that could use more resources. "We need more disk space" does not impress people with signing authority, but "We had eight hours of system downtime last month because I was playing partitioning games that could be solved by purchasing another $3,000 of disk'' usually does.

What should go into your logs? Ultimately, anything that you think is useful. Certainly, details on any problems you solve, including the syntax for any commands you're likely to forget before the problem re-occurs. If you have a great idea, or the germ of a great idea, but no time to implement it, put it in your log-- don't rely on a Post-It to be there in a month. Watching the growth of some files on your system? Put the checkpoint data into your log file. If you are angry at a user or your management, vent your frustration into your log and, after a suitable cooling off period, go back an look at what you wrote. Maybe it won't seem so important later, or maybe you can spot a way to correct the situation that you couldn't see in the heat of the moment. The best advice is to start off being as verbose and compendious as possible in what you log. Over time you will see what is useful and what is not. Also, there's the human tendency to become lazier over time: if you get in the habit early of logging large amounts of data, then you'll still be logging something when laziness sets in.

Your log should be updated often enough to keep it complete. If a nightly braindump works better for your schedule, then so be it. Other people find they need to update their logs more frequently to capture commands and output that they need to solve future problems. Sometimes it is simply impossible to update your logs as frequently as you would like (because you are spending all of your time fighting fires, for example). Update your log as soon as you can get out of crisis mode.

Perhaps the most divisive issue is whether logs should be kept electronically or in a notebook or other paper medium. On the plus side, it's easier to capture command output into an electronic logbook. Also most System Administrators spend large amounts of time in front of a keyboard, and so keeping a log on-line is often more natural than shifting gears to pen and paper. It's easy to run *grep* (or other text manipulation tools) against an electronic logbook, and rather more labor intensive to do it against paper. On the minus side, it's difficult to use your electronic logbook to help you solve the problem of getting the host which contains your logbook back on-line. Also, many System Administrators work in environments where they don't have consistent access to a single machine to keep their logs on, but in fact travel to many sites during a single day or a single week. For these people, a notebook and pen may be the only solution.

This ability to easily take your logs anywhere may be the biggest point in favor of hard-copy logging. It's also easier to doodle or draw pictures for yourself in a notebook when you're trying to work through a problem: paper logs can be much more free-form in style and content. One negative is constantly having to carry your logbook with you. If your logs are compendious, it may become difficult to physically carry much history around. Paper logs don't have an easy backup mechanism and can often be difficult to search for a small piece of information. If you do keep paper logs, be sure to use ballpoint or some other non-water soluble ink. Nothing is more depressing than having a logbook turned into an abstract watercolor design by a sudden rainstorm or unexpected trip into the sink.

Most importantly, choose a format and style of logging that is useful to you. If your log becomes spotty and incomplete, or if you only enter data into the log and never review your logs later, then you must ask yourself why you are spending time keeping a log at all.

PLOD User Interface

The user interface for PLOD looks a great deal like Berkeley mail. It is a simple text interface that many System Administrators are familiar with. For those who prefer a different interface, Paul Foley (*paul@ascent.com*) has developed an Emacs mode for PLOD which contains a number of nice features. Paul's *plod.el* file is distributed with PLOD or may be obtained from him directly.

The simplest use of PLOD is making one-line log entries from the command line:

```
host% plod A one-line log entry.
```

Typically, however, the user will want to make a multi-line entry. Simply invoke PLOD with no arguments: a date/time stamp will be printed, and the user may enter any number of lines of logging information, ending their entry with a bare period.

```
host% plod
09/13/93, 15:56 --
Here is an example of
a multi-line log entry
.
(eot)
host%
```

A number of tilde-escapes are supported in multi-line mode. For example, *~r filename* will append the contents of a file to the current log buffer:

```
host% plod
09/13/93, 16:00 --
~r .cshrc
.cshrc: 56 lines
(continue composing note)
Some more text could
go here, as desired.
.
(eot)
host%
```

A complete listing of tilde-escapes can be obtained by typing *~h* or *~?* while in multi-line mode.

By default, log entries are stored in encrypted format in the directory *.logdir* in the user's home directory. Individual log file names are four characters long in the format *YYMM*, and so a new file is started at the beginning of every month. Log entries within each file are separated by five dashes (-----) on a line by themselves. The dashes are followed by the time/date stamp for the log entry and the text of the entry itself. Note that the decision to encrypt, the location of the logging directory, the name of the log file (and therefore the cycle time before beginning a new file), and even the format of the date/time stamp are completely customizable (see the next section).

Several command line options are provided to allow users to edit or review old log files. These command line options also correspond to tilde-escapes in multi-line mode (i.e., the command line option *-E* is equivalent to the tilde-escape *~E*). The command *plod -P* will invoke the user's *PAGER* on the current log file (decrypting the file if necessary). Similarly, *plod -E* (*plod -V*) invokes the user's *EDITOR* (*VISUAL*) on the current log file. *plod -C* will simply *cat* the current log to the standard output, which can be useful for pipelines like *plod -C | lpr*.

All of the above command line options will optionally accept the name of an older log file, plus encryption key as needed. The default encryption key is *pl<yr><mn>od*, so to review the log file from August, 1993, the user would type:

```
host% plod -P 9308 pl938od
```

Note that the default file name is always four characters long, since the month portion is zero-filled, but the default encryption key does not use a zero filled month.

PLOD Customization

Customizing PLOD can be as simple as changing an environment variable or as complex as adding a new tilde-escape. Since PLOD is written in Perl, it is also easy to customize by modifying the script directly. This is not recommended, however, since modifications will have to be reapplied to each new release. PLOD has a rich set of hooks for customization, so please evaluate them carefully before modifying the distributed code. The author is always happy, however, to incorporate bug fixes or useful new features.

A number of defaults have been hard-coded into the PLOD program itself. When executed, PLOD will look for a system-wide */etc/plodrc* file for machine or site-dependent configuration information. After the */etc/plodrc* file has been evaluated, the user's environment variables are searched for personal configuration choices. Finally, PLOD checks the user's home directory for a *.plodrc* file which may contain additional or PLOD-specific configuration information.

PLOD recognizes the common *EDITOR*, *VISUAL*, *PAGER*, and *HOME* environment variables. The *LINES* environment variable controls how long the output of various tilde-escapes must be before *$PAGER* is invoked to provide output by screenful. This is somewhat like the *set crt = n* option in BSD mail.

A number of environment variables are used to control the location of various files that PLOD creates or manipulates. *LOGDIR* specifies where the log files are stored and *LOGFILE* is the name of the file in that directory. If *LOGFILE* begins with a slash (/), then *LOGDIR* is ignored and *LOGFILE* is taken to be the absolute path to the current log file. PLOD attempts to trap SIGKILL and SIGQUIT, and aborted logs are dropped into the file *$HOME/$DEADLOG* (unless, again, *DEADLOG* is an absolute pathname). *TMPFILE* is the absolute pathname of the scratch file PLOD uses for various operations. This file is always removed once the operation is completed.

CRYPTCMD is the absolute pathname of the command used to encrypt log files. The default is */bin/crypt*, which is not in the least secure but does provide protection against casual browsing. If encryption is not desired, simply set *CRYPTCMD* to be null. *KEYVAL* is the encryption key to be used by *CRYPTCMD*.

Finally, PLOD recognizes the *STAMP* environment variable to customize the format of the date/time stamp associated with each log entry. Note that it is generally inadvisable to put a line like

```
setenv STAMP "'date +%m/%d/%y, %H:%M'--"
```

in your *.cshrc* or other shell startup file, since this would imply the same date/time stamp for all log entries during the life of the shell. It is better to customize this variable in the */etc/plodrc* or *~/.plodrc* files, which are re-evaluated at each execution of PLOD.

Unlike most UNIX configuration files, the */etc/plodrc* and *~/.plodrc* files are evaluated as Perl code. Thus, if you wanted to customize the *STAMP* variable in your own *~/.plodrc*, you could write:

```
$STAMP = sprintf("%02d%02d%02d %02d:%02d",
        (localtime)[3,4,5,2,1]);
```

(line broken for display purposes) which would give you something like *DD/MM/YY HH:MM* in the European fashion.

While the requirement that the */etc/plodrc* and *~/.plodrc* be correct Perl syntax may be a barrier to novice users, it opens huge vistas of customization possibilities. Customization through environment variables is still an acceptable option for the non-Perl oriented.

Perhaps the nicest feature of evaluating the configuration files as Perl code is allowing users to extend PLOD by adding their own escape sequences. PLOD maintains all escape sequences as a global array, *%funcs*, of function pointers (actually, since Perl does not currently support the notion of a pointer, type globs are used). Functions are indexed in the array by the letter of their tilde-escape. For example, Figure 1 shows the definition of the *~!* escape which executes a shell command and then returns to PLOD. PLOD automatically passes any arguments following a tilde-escape to the appropriate function. The entire argument list is passed in as a single string which may have to be broken up if the function needs to process its arguments one at a time.

The scalar *$bang* is an optional descriptive message used by the on-line help function invoked with *~h* or *~?*. As Figure 2 shows, the function simply extracts a sorted list of all escape sequences defined in the *%funcs* array, and prints each one followed by the descriptive string referenced by the associated type glob. This example shows that it is trivial to bind the same function to multiple tilde-escapes. Note also the use of the *LINES*, *PAGER*, and *TMPFILE* customization variables.

In addition to the *%funcs* array, two other global data structures are available for user-defined escape sequences to manipulate. The list *@lines* contains the lines of text the user has entered for the current log entry. The first element of the list is always the date/time stamp. Figure 3 shows the code for the *~a* escape which will append the contents of the current log entry to a file.

The other data structure available is the list *@buffer*. This contains the last long Perl fragment entered using the *~M* escape. This escape sequence allows users to enter multi-line Perl fragments on the fly and have them incorporated into the current invocation of PLOD. It is possible to enter code that will manipulate *@lines*, and therefore the contents of the current log entry. It is even conceivable that such a code fragment could modify its own image in *@buffer* and produce some horrible hack of self-modifying code. This is not recommended.

Creating one's own escape sequences in not the limit of the power of this customization mechanism. Perhaps the */etc/plodrc* file could contain code to evaluate a site-wide */usr/local/etc/plodrc* before any machine-specific customizations in */etc/plodrc*:

```
sub bang {
   local($cmdline) = @_;
   system "$cmdline";
   print "(continue composing note)\n";
}
$bang = "cmdline\tExecute system command and return";
$funcs{'!'} = *bang;
```

Figure 1: Code to implement escape mechanism

```
sub helpuser {
  $long=(scalar(keys %funcs)>=$LINES) && open(TMP, ">$TMPFILE");
  for (sort keys %funcs) {
    *info = $funcs{$_};
    if ($long) {
       print TMP "~$_ $info\n";
    }
    else { print "~$_ $info\n"; }
  }
  if ($long) {
    close(TMP);
    system("/bin/cat $TMPFILE|$PAGER");
    unlink "$TMPFILE";
  }
}
$helpuser = "\t\tPrint this message";
$funcs{'h'} = *helpuser;
$funcs{'?'} = *helpuser;
```

Figure 2: On-line help function

```
if (-e "/usr/local/etc/plodrc") {
   eval { do "/usr/local/etc/plodrc"; };
   die "*** Error in " .
      "/usr/local/etc/plodrc:\n$@" if $@;
}
# do machine-specific customizations here
```

Further chicanery is left as an exercise to the interested reader. If you develop something good, please inform the author.

Conclusion

PLOD is a living breathing entity which owes many of its features to the ideas of others. The author greatfully acknowledges the contributions of David W. Crabb (crabb@phoenix.Princeton.EDU), John Ellis (ellis@rtsg.mot.com), Mike Lachowski (mlachow@erenj.com), Eric Prestemon (ecprest@pocorvares.er.usgs.GOV), Erik E. Rantapaa (rantapaa@math.umn.edu), and James Tizard (james@ringo.ssn.flinders.edu.au). The original idea for PLOD came from a conversation with Bill Mendyka (mendyka@dg-rtp.dg.com) at LISA VI.

The author would not dare to suggest that PLOD is the logging tool for everybody. A goodly number of people find it useful. You may choose to keep a journal via some other mechanism, but always keep a record of the work you do. The payoff may be infrequent, but is often enormous.

PLOD v1.6 has been recently posted to comp.sources.misc and comp.lang.perl. The comp.sources.misc newsgroup is archived at many FTP sites. Scripts posted to comp.lang.perl are archived at coombs.anu.edu.au. In addition, a shar file is available via anonymous FTP from gatekeeper.imagen.com in the directory /pub/plod. If all else fails, the author will be happy to satisfy email requests for the current version.

Author Information

Hal Pomeranz is a victim of being in the right place at the right time. A grant allowed his undergraduate institution to purchase a state of the art (at the time) workstation network for the amusement of the student hackers who were nominally supposed to keep things running. This was apparently enough of a credential to launch a career that has included System Administration stints at AT&T Bell Labs and the NASA Ames Research Center. Hal Pomeranz is the author of the fabulously overlooked *Perl Practicum* column for the USENIX *;login:* Magazine. His current email address is pomeranz@aqm.com.

```
sub appendfl {
  local($file) = @_;
  if (!open(OUTP, ">> $file")) {
    warn "*** Could not append to file $file\n";
    return;
  }
  print OUTP @lines;
  close(OUTP);
  print "Wrote ", scalar(@lines), " lines to file $file\n";
  print "(continue composing note)\n";
}
$appendfl = "file\t\tAppend contents of buffer to file";
$funcs{'a'} = *appendfl;
```

Figure 3: Implementation of ~a escape

How to Keep Track of Your Network Configuration

J. Schönwälder & H. Langendörfer – TU Braunschweig, Germany

ABSTRACT

In this paper we present extensions for the *Ined* network editor allowing us to discover the structure of an IP network automatically. The discovering algorithm is based on an active probing technique that fits well with our interactive editor. We have chosen a multi-threaded approach to minimize response times which seems reasonable in fast networks but may fail when run over slow serial links. A set of utilities have been designed to lay out the discovered network based on additional information found in the Domain Name System.

Introduction

Maps showing the network configuration are very important documents for a variety of tasks. This motivated us to design and implement a domain specific editor called *Ined* that is specialized for drawing and maintaining network maps [7].

Ined provides all necessary editing features to draw and maintain network maps. Maps may be composed of node, network and link objects. Link objects represent connections between two node objects or a node and a network object. A large map can be structured into a hierarchical map using group objects that can contain other *Ined* objects. The editor has a programming interface allowing scripts written in the Tool Command Language (tcl) [3] to control objects shown inside of the editor. This interface enables users to write macros for frequently used command sequences. It also serves as an interface for scripts performing network management tasks.

Once we had a proper drawing program, we found it a very lengthy and tedious task to enter all the network information into the editor. Another problem is to keep network maps current since there are frequent changes in our network with about 1.300 registered hosts. So we decided to design and implement extensions for the *Ined* editor that will assist us in these tasks. The result of our work is an IP network discovery tool and a set of utilities simplifying the final layout process.

Design Decisions

When designing a network discovering program, you have to make some very important design decisions. First you have to determine the network layer on which to focus. It is much easier to design a smart discovering program if you have a clear understanding of what information should be discovered. We focus on the IP layer since this is the most frequently used protocol in our environment and it is supported by nearly every device on the network.

Next you have to decide if you will use passive monitoring techniques or if you will actively probe the network to gather information about its internal structure. The first alternative provides no direct way to guide the discovering process since you can only observe the current packet exchanges. Therefore it may take a long time gathering enough information to derive a complete picture of your network. Another problem with passive techniques is that you must have monitoring devices attached to all your subnetworks (e.g., Ethernets) and that the implementation under the UNIX operating systems requires a network interface tap which is not available on every machine.

The main advantage of passive monitoring techniques is that they put no additional load on the network itself. Using active probing techniques you must be aware of the additional load your network must carry. If your network is sensitive in this respect you will probably be in favor of passive techniques. On the other hand, active techniques can be directed easily. Therefore it is possible to minimize the time needed to get a complete map of your network.

Since our program should support an interactive editor we have decided to use an active probing technique. We are taking an aggressive approach in order to minimize the total time needed to discover a complete network. This is reasonable on fast networks but may fail when run over slow serial links.

The third design decision concerns the information sources available to the discovering program. There are two realistic choices. You can either use a network management protocol (e.g., SNMP [12]) to gather information, or you can try to use features of protocols handling ordinary network traffic for your purpose.[1] Using a management protocol seems attractive at first sight but it requires that most

[1] Sure, you can invent your own protocol. But it is very unlikely that it will be installed on every machine in a large network.

machines on your network are able to respond to management protocol requests. The second alternative is more likely to work with a large set of different devices but makes the analysis of the acquired data difficult.

The discovering algorithm described in the next section mainly uses the ICMP protocol [5] to discover the structure of a network. This has the advantage that every device on the IP network can be detected since every IP implementation must support the ICMP protocol. Additional information is retrieved from the Domain Name System (DNS) if available.

How Discovering Works

The network discovering algorithm is divided into nine steps. The first four steps send probing packets to gather data while the remaining steps are used for data analysis. The algorithm assumes an implementation which will be able to send request packets in a round-robin fashion while waiting for outstanding reply packets to arrive or timeout. This way the total time needed for a given address space remains constant, regardless of the number of responding hosts.

1. Determine IP addresses in use by sending ICMP echo request packets to every address of the given address space. We use a sequential approach since directed broadcast ICMP echo requests tend to cause lots of collisions or even broadcast storms due to errors in some IP implementations. Some IP router even remove incoming broadcast packets reducing the usefulness of directed broadcasts.
2. Trace the routes to all IP addresses discovered in step one using the Van Jacobsen algorithm [10]. The traces are stored for later analysis. Gateways showing up during the traces are added to the IP address list.
3. Determine the network mask for each IP address using the ICMP mask request. The network masks are saved for later analysis.
4. For every IP address, send an UDP packet to an unused port and save the IP address contained in the port unreachable reply packet. Some multi-homed devices respond to an incoming packet addressed to one of the remote interfaces with a packet containing the IP address of the incoming interface. The returned address is stored for later analysis.
5. Identify networks and subnetworks. Class A, B and C networks are easily recognized by examining the IP addresses. Subnetworks are a bit tricky. First collect all potential subnets based on the netmasks returned in step 3. Afterwards, check if the majority of all members of a potential subnet has reported a suitable netmask. This two level approach is needed to handle incorrect netmasks properly.
6. Identify multi-homed machines based on the traces and the address contained in the port unreachable reply packet of step four. Comparing the Domain Names of the IP addresses gives additional hints to multi-homed machines since the Domain Name Service often contains records with the same name for each interface of a multi-homed host.
7. Connect IP addresses to the networks identified during step six. Gateways are connected based on the traces. We found that gateways often return a different IP address when being traced. Therefore we skip the last hop of traces that end at a gateway machine.
8. Merge the IP addresses of multi-homed machines. This step cleans up duplicate information stored for each interface of a multi-homed machine.
9. Download the current map from the *Ined* editor and add all discovered objects to the map that do not exist yet. Hosts and gateways are mapped to node objects, networks to network objects and IP interfaces to link objects.

The above algorithm can be used to discover routing traces by initializing the list of IP addresses and starting the algorithm at step two. This is a very nice extension to the traceroute program [10]. The resulting map lets you easily identify machines where branches join that are important for your site.

Layout

The output of the discovering algorithm is a set of nodes, networks and links placed on a grid. Some separate tools help us to lay out the discovered topology. The separation of these tools from the discover process allows us to use them even when entering network maps interactively.

The layout tools are divided in a set of commands that are specific to IP networks and a set of commands that are of general use. The second set includes commands to manipulate the selection of the editor and the text shown inside the labels.

The IP specific commands mostly deal with the Domain Name Service (DNS). They fill empty attributes of *Ined* objects with DNS information or set icons automatically based on HINFO records. A table of regular expressions maps HINFO records to icon names.[2] Another IP specific command removes identical domain name endings from host names. This command shortens labels considerably resulting in more readable maps.

The most important layout utility rearranges icons on the map. We have chosen a simple method that places hosts connected to a network on a grid

[2]Many site administrators have their own naming convention although there are some guidelines and Official Machine Names in the Assigned Numbers RFC [9].

around the network. Hosts with more than one interface are placed on the border of a cluster. Another command allows us to group all nodes of a network having exactly one interface. Used in conjunction with the previous command, we get a map focusing on networks and their gateways.

The map in Figure 1 shows the networks owned by the computer science departments of our University. All 6 subnetworks are linked by the backbone network (134.169), the cisco router and the serial line network (134.169.247). Subnet 134.169.33 is shown as an expanded group. The icons have been arranged around the network and the domain name endings have been removed.

Implementation

The network discovering algorithm is implemented as a script written in the Tool Command Language (tcl). We have extended the standard tcl interpreter with a command allowing us to send ICMP packets. The use of a script language has the advantage to allow modifications and customizations at very low cost.

The ICMP module is implemented as a separate process called ntping. ntping runs setuid root since sending and receiving ICMP packets requires access to raw sockets. ntping creates a new job for each destination address read from stdin and processes these jobs in round-robin fashion. A tcl command hides the communication with ntping from the tcl programmer. Figure 2 shows the options of the tcl icmp command.

```
icmp [-size <b>] [-delay  <ms>]
     [-retries <n>] [-timeout <s>]
     [-ttl <n>] [-trace <n>] [-mask]
     <hosts>
```

Figure 2: tcl icmp command

The -ttl and -trace options are used to send an UDP packet to an unused port with a time to live set to <n>. The difference between these two options is that -trace always returns the IP address of the destination host while -ttl returns the IP address actually contained in the reply packet. The -delay option can be used to force a minimum delay between two successive packets. Delays have proven useful to reduce load on intermediate nodes like bridges or routers. The implementation of the delay option uses a loop with calls to gettimeofday for short intervalls since the usleep library call does not handle short intervals reliable on many machines.[3]

[3]Berkeley Unix uses signals to implement usleep. This causes a series of system calls which usually take more time to complete than the requested interval if the interval is short. The minimal delay obtained using usleep on a SparcStation was 20ms.

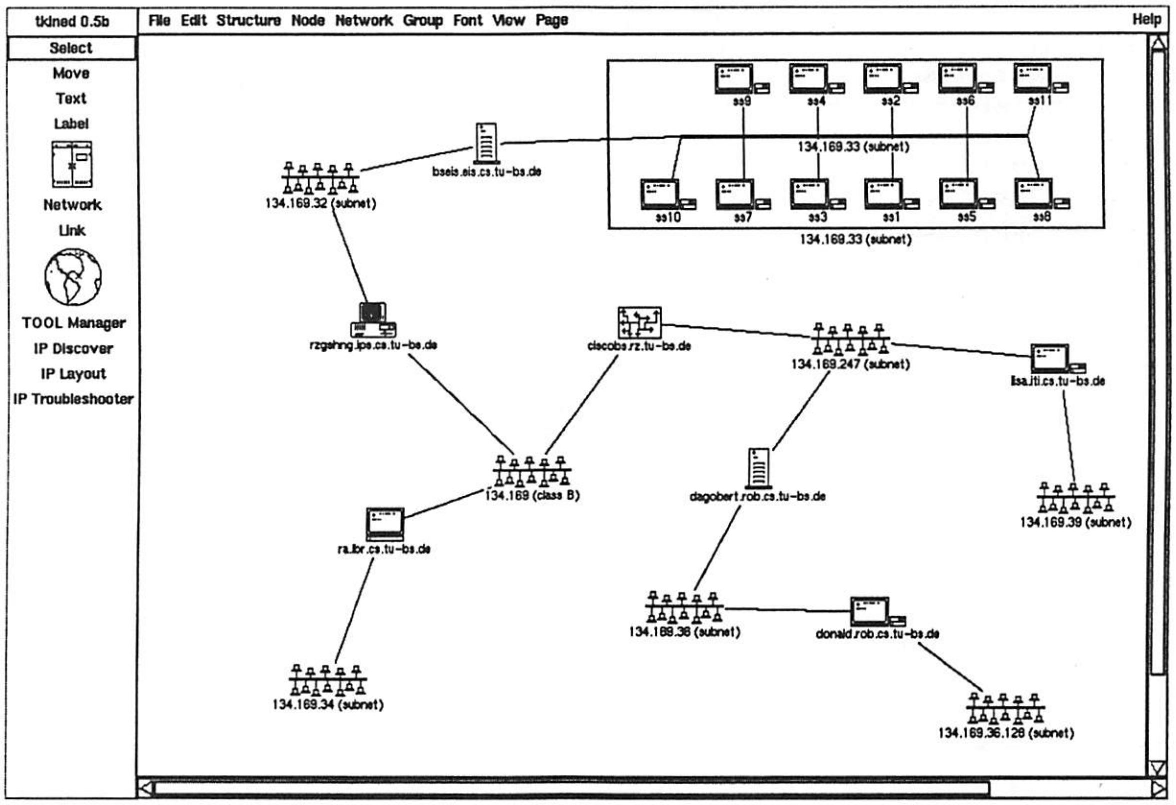

Figure 1: A sample Map

The efficiency of the script can be controlled by adjusting the parameters shown in table 1. We found the default values appropriate on our network (mainly Ethernet networks). You should increase the delay and timeout times if you have slow serial lines in your environment. The maximal length of a route only affects step two of our algorithm. Increasing this value is useful if you want to use the script to discover routes on a wide area network.

Parameter	Default
number of retries	3
timeout [s]	3
delay [ms]	5
max route length	8

Table 1: Discover Parameter

Table 2 shows some measurements for class C and class B networks. For each step of our algorithm, the number of discovered devices and the completion time is given. The most time consuming part is the initial pass to discover IP addresses in use. The current implementation breaks a class B address space in 255 class C like networks and processes these networks sequentially. Removing this limitation will increase the number of active threads which will reduce the overall time needed for class B networks.

Related Work

Another network discovering system called Fremont has been build by Wood, Coleman and Schwartz [8]. Fremont uses eight different explorer modules to discover network characteristics. All information gathered by these explorer modules is stored in a journal that is used to direct the exploring process and to uncover network problems.

The first difference of our approach and the Fremont systems is that we do not use passive monitoring techniques. Passive monitoring of network traffic only extracts information about the subnet the monitoring host is directly attached and it requires a long observation period. Since we are trying to discover network characteristics fast, passive monitoring techniques did not seem attractive to us.

The Fremont systems takes care not to load the network with probing packets. This seems reasonable since the explorer of the Fremont system makes regular passes through the network. Our system is meant to operate in fast local area networks, and is designed as a network analysis utility to be run occasionally. Hence an active probing algorithm seems applicable.

We further believe that gathering of information about the structure of subnetwork layers is best done by the hardware that connects these subnetworks (e.g., bridges and routers). Since more and more of these devices are capable of running SNMP, we are convinced that supporting SNMP and ICMP suffices.

Future Work

Our current work on the discovering tool will add support for SNMP. This will allow us to detect network configuration below the IP layer by querying hosts, gateways and bridges for their address translation tables. P. H. Kamp [1] has done some work to implement SNMP extensions for tcl supporting an asynchronous interface to send and receive SNMP packets. We expect to get the SNMP discover tools as fast as ntping using this asynchronous interface.

Routing tables retrieved via SNMP can be used to discover routes between two remote hosts. We will implement an algorithm similar to that of xpath [11] once we have finished our SNMP implementation.

Another activity is the integration of a configuration database in order to store information about discovered devices. The database is comparable to the journal server of the Fremont system but

Step	Class C		Class B	
	Number	Time [s]	Number	Time[s]
ICMP echo request	51	10	610	2453
Trace routes	51	9	610	289
ICMP mask request	52	4	610	24
Get remote interface address	52	3	610	15
Identify networks	2	2	5	32
Identify multi-homed machines	1	1	23	7
Connect hosts to networks	97	4	1004	260
Merge multi-homed machines	1	0	23	1
Create *Ined* objects	107	8	1219	102
Time to complete		41		3181

Table 2: Speed of the Discovering Script

may be used for general system management tasks as well [6].

Conclusions

In this paper we have presented a network discovering tool and how it provides an efficient method to get a map of your current IP network setup. The discovering mechanism is designed to operate on fast local area networks that wont suffer from the aggressive probing algorithm used. First experiments show that the approach is working fast and reliable. As noted by Wood, Coleman and Schwartz, discovering IP networks using ICMP packets is a very attractive approach. Other protocols that can be used for exploration are less widespread and they often require additional knowledge (like community names for SNMP).

Acknowledgements

We would like to thank Erik Schönfelder for the implementation of ntping. Also many thanks to Stefan Petri for his helpful experiments with directed broadcast pings.

Availability

The *Ined* editor has been re-implemented using the tk toolkit [4] and is now called tkined. It is available by anonymous ftp from ftp.ibr.cs.tu-bs.de in the directory /pub/local. You will find the source of our tcl interpreter scotty which includes ntping and the script described in this paper in the same directory. You are invited to join the tkined mailing list. Send your request message to tkined-request@ibr.cs.tu-bs.de

Author Information

Jürgen Schönwälder received his diploma degree in Computer Science from the Technical University of Braunschweig. He is now a member of the research staff at the Institute for Operating Systems and Computer Networks. His interests include system administration, network management, distributed systems and network security. Reach him via Mail at TU Braunschweig, Bültenweg 74/75, D-38106 Braunschweig, Germany. Electronic Mail should be sent to schoenw@ibr.cs.tu-bs.de.

Horst Langendörfer is Professor at the Technical University of Braunschweig since 1981. His research interests include operating systems, distributed systems, performance analysis, computer networks, network management and network security.

References

[1] P. H. Kamp. *tcl_snmp – SNMP interface for tool command language.* March, 1993.

[2] R. Lehman, G. Carpenter, and N. Hien, *Concurrent Network Management with a Distributed Management Tool.* Proc. of LISA VI, pages 235-244, 1992.

[3] J. K. Ousterhout. *TCL: An Embeddable Command Language.* Proc. Winter USENIX Conference, pages 133-146, 1990.

[4] J. K. Ousterhout. *An X11 Toolkit Based on the TCL Language.* Proc. Winter USENIX Conference, pages 105-115, 1991.

[5] J. Postel. *Internet Control Message Protocol.* RFC 792, September 1981.

[6] J. Schönwälder and H. Langendörfer. *Administration of large distributed UNIX LANs with BONES.* Proc. SANS II, Arlington, April 1993.

[7] J. Schönwälder and H. Langendörfer. *INED – An Application Independent Network Editor.* Proc. SANS II, Arlington, April 1993.

[8] D. C. M. Wood, S. S. Coleman, and M. F. Schwartz. *Fremont: A System for Discovering Network Characteristics and Problems.* Proc. USENIX Winter Conference, pages 335-347, January 1993.

[9] J. Reynolds and J. Postel. *Assigned Numbers.* RFC 1340, July 1992

[10] V. Jacobsen. *Traceroute Software.* Lawrence Berkeley Laboratories, December, 1988.

[11] A. Leinwand and J. Okamoto. *Two Network Management Tools.* Proc. Winter USENIX Conference, pages 195-205, 1990.

[12] J. Case, K. McCloghrie, M. Rose, S. Waldbusser. *Introduction to version 2 of the Internet-standard Network Management Framework.* RFC 1441, April 1993

Forecasting Disk Resource Requirements for a Usenet Server[†]

Karl L. Swartz – Stanford Linear Accelerator Center

ABSTRACT

Three years ago the Stanford Linear Accelerator Center (SLAC) decided to embrace netnews as a site-wide, multi-platform communications tool for the laboratory's diverse user community. The Usenet newsgroups as well as other world-wide newsgroup hierarchies were appealing for their unique ability to tap a broad pool of information, while the availability of the software on a number of platforms provided a way to communicate to and amongst the computing community. The previous way of doing this ran only on the VM mainframe system and had become increasingly ineffective as users migrated to other platforms.

The increasing dependence on netnews brought with it the requirement that the service be reliable. This was dramatically demonstrated when the long-neglected netnews service collapsed under the load of the traditional fall surge in Usenet traffic and the site was without news service for a week while an upgraded system was installed. One result of that painful event was that efforts were made to forecast growth and the accompanying hardware requirements so that equipment could be acquired and installed before problems became visible to the users.

This paper describes the major on-disk databases associated with news software, then presents an analysis of the storage requirements for these databases based on data collected at SLAC. A model is developed from this data which permits forecasting of disk resource requirements for a full feed as a function of time and local policies. Suggestions are also made as to how to modify this model for sites which do not carry a full feed.

Why NetNews? Why Usenet?

The Stanford Linear Accelerator Center (SLAC) is a medium-sized national research laboratory. The lab's primary missions are research in elementary particle physics and development of new techniques in particle accelerators. These are large projects that often involve international collaborations and a diverse user community. This can present a formidable problem for communication with and amongst users, from discussions on the design of new experiments to progress reports on current experiments, as well as the more mundane but equally necessary announcements of network or server outages. There is also a tremendous need to stay in touch with what other researchers are doing.

In the eighties, the majority of computer users at SLAC logged onto an IBM mainframe running VM. The default user profile on VM brought up a VM News session upon login, where announcements could be made. Most users would see them since they logged into VM regularly. Discussions were handled by mailing lists maintained by LISTSERV, plus a home-grown VM conferencing system named CONSPIRE. (VM News was also developed at SLAC.) BITNET mailing lists provided the main contact with researchers at other sites.

There was also a smaller, though still sizable, group of users who used VAX/VMS systems most of the time, if not exclusively. These users tended to be excluded from VM News and from CONSPIRE, though they did make use of BITNET. DECnet mail with other physics sites was also available. This communications block posed some problems but was tolerated at the time.

Over the past few years the computing environment became far more diversified. Unix workstations began to appear, while the PCs, Macintoshes, and Amigas grew powerful enough that their users had dwindling need for VM. The number of users who did not use VM on a regular basis, if at all, increased until the VM-based communication model began to fail completely. A new model was needed that would permit users to read announcements and participate in discussions from whatever platform they were accustomed to using. The multitude of platforms made another home-grown solution undesirable. The software which supports Usenet (referred to hereafter as netnews software to distinguish it from the network itself), with ready availability of NNTP-based readers for a variety of platforms, seemed to solve the problem except for VM. The discovery of the PennState VM NetNews system completed the solution.

[†]This work supported by the United States Department of Energy under contract number DE-AC03-76SF00515, and simultaneously published as SLAC PUB-6353.

B News and NNTP software was built and installed on a Sun fileserver which had some spare disk space, and the PennState NetNews software, with NNTP software from Queen's University, was installed on VM.[1] Once the bugs were worked out of this system, NNTP-based readers were acquired for other platforms, often with help from interested users. Meanwhile, a local *slac* newsgroup hierarchy was being populated with new groups and users were being introduced to the new system.

There was also a great deal of interest in non-local groups, of course, i.e., Usenet. After a great deal of debate over proper use of government-funded equipment, the appropriateness of censorship in an academic/research community, and the feasibility of determining just what was and was not appropriate, it was decided to carry all groups and assume a mature and responsible user community.

Netnews flourished and mostly solved the communication problem, but created a new problem in that users came to expect it to be reliable. By early 1992 the ever-increasing growth of Usenet traffic had begun to severely strain the resources of the Sun which was trying to run netnews while also handling file service and a variety of other tasks. Spool areas would overflow and the now-obsolete B News software would collapse, causing substantial delays and user ire. The expiration noose would be tightened another notch, staving off disaster for a bit longer but also aggravating already irate users as articles expired before they could be read.

Work was begun to determine what equipment was needed to support netnews for the next few years, and to define requirements in the form of minimum expiration times. Eventually a SPARCserver 2 with 4.5 GB of disk space was ordered, arriving mere days too late to avert the collapse of SLAC's netnews service from the traditional September surge in Usenet traffic. While painful at the time, the degree of pain made it clear just how critical netnews had become to SLAC.

Popular reference books on managing Usenet are notably silent on the matter of forecasting resources. [1, 2] Therefore, a study of Usenet growth was begun and has continued, so that future growth needs can be anticipated and handled before another crisis occurs. This paper documents the current state of this ongoing study.

Organization of a Netnews Server's Data

A netnews server is composed of a number of databases, several of which have the potential to require a substantial amount of disk space and which fluctuate in size as a function of news activity. The three primary databases are the articles themselves, the history file, and the active file. Ancillary structures include thread databases, incoming and outgoing spool areas, and log files. While the following description is based on a Unix system running the C News software, most structures will likely be similar on other systems.

The article database is by far the largest portion of a news system. Unix's hierarchical directory structure is a handy analog to the newsgroup hierarchy, so the software uses a simple lexical mapping of newsgroup names into directory names ("." is mapped to "/") with each article stored in a separate file. For example, article 42 in group *news.announce.important* is stored as `news/announce/important/42` within the spool directory (often `/usr/spool/news`). Cross-posted articles are handled by links. These are normally hard links, though there is some support for falling back to symbolic links if a hard link fails.[2]

Storing each article in a separate file implies a tremendous number of relatively small files. If one eschews the apparently lightly tested symlink support – an option not likely to be available for large netnews servers for much longer – the use of links for cross-posted articles further implies that this entire database must reside within a single disk partition. Unfortunately, the combination of cross-posts, cancellations, varying expiration times for different newsgroups, and the common desire to be able to use grep and other Unix tools on the article database make an alternative implementation difficult and thus unlikely in the near future. (This is a recurrent thread in the *news.software.** newsgroups.)

The history file, typically located in `/usr/lib/news/history`, records the article ID of each article seen recently by the news system along with the time it was received, any explicit expiration time, and, if the article has not yet expired, a list enumerating the newsgroups the article appears in and its sequence number in each group. All of this is stored in a simple, albeit large, file with one article mentioned per line. In order to speed lookups by article ID, an index is maintained, typically using dbz.

The last of the primary news databases is the active file, typically `/usr/lib/news/active`, which lists each newsgroup known to the news system and, for each group, the range of article

[1]VM NetNews is a full news system, not an NNTP-based reader. There are now several NNTP readers available for VM, including a port of rn. These are being investigated, with the intention of consolidating on a single, UNIX-based netnews server.

[2]The documentation for the February 1993 C News Performance Release describes symlink support with uninspiring phrases like "half-hearted code" and "this has not been tested much recently." [3] There appears to be more confidence in the symlink support in INN 1.4. [4]

numbers currently active along with the moderation or other status of the group. Like the history file this is a simple file. It is small, however, and so will not be considered further in this paper.

The largest of the ancillary structures is one or possibly several thread databases, which allow newsreaders to present related articles in a logical order, rather than in whatever order they happened to arrive on the netnews server. One example formed the heart of the *trn* newsreader. [5] This required about 5% of the space required for the articles, [6] a not insubstantial usage of space which was compounded by the development of several incompatible solutions for other newsreaders. Fortunately, a standard thread database is being adopted in the form of Geoff Collyer's Overview or NOV (News OverView). Unfortunately, the generality required to support the varying needs of different newsreaders, current and future, demands more space – approximately 10% of the space consumed by articles. [7]

Whatever threading mechanism is used, a separate file is maintained for each newsgroup, with configuration options to choose where these files are stored. One choice is to store them amongst the articles for the newsgroup, e.g., *News OverView*'s database for the *alt.sewing* newsgroup might reside in `/usr/spool/news/alt/sewing/.overview`. Since the article database is already quite large, and possibly constrained to a single partition, a more prudent choice is to use a separate directory tree, e.g., *trn* might store its database for the same newsgroup in `/usr/spool/threads/alt/sewing.th`.

The incoming spool area contains (batches of) articles that have been received but not yet processed by the news system. As long as the netnews server processes new articles expeditiously, this requires only modest amounts of space. (In fact, INN handles incoming NNTP traffic entirely within memory, making its space requirement zero.)

Outgoing spool space is even smaller, since it contains no articles, just references to them in the form of article IDs and/or pathnames, and these hopefully don't stick around very long. For transport mechanisms other than NNTP, e.g., uucp, space will also be required for batches queued for other systems. This paper doesn't consider this requirement, as most large sites are using NNTP these days.

Finally, there are the various log files, stored under `/usr/lib/news`. The main log records each article received by the system along with a timestamp and an indication of what was done with the article. Normally this log is restarted each night, with a few days worth of old logs kept around. There is also `errlog`, which hopefully is empty or nearly so, and perhaps `batchlog`, which should also be quite small.

Factors Influencing the Size of Databases

The size of the various databases described previously depends on a number of factors. Some, such as article life before expiration, are controlled by the news administrator. Others depend entirely on external influences and, over time, may vary.

By far the most ravenous consumer of disk space in netnews is the article database, so it makes sense to scrutinize most carefully those factors which affect the size of this database. The major ones are the average size of an article, and the rate at which new articles are received. Filesystem overhead can be significant as well, but this is a fairly simple function of the other two factors and, of course, the characteristics of the netnews server's system software.

Other factors include the size of article IDs and newsgroup names, cancellation rates, and cross-post rates. These play a comparatively minor role in the growth of disk consumption by netnews, as they are only slowly changing, and they have little effect on the article database. A cursory study of these factors – sufficient to establish some baseline data – is all that is attempted here.

Number of Articles Received per Week

From the beginning SLAC's netnews server was configured to keep one week's worth of old logs, and to generate a weekly traffic report based on these logs.[3] Many sites in the San Francisco Bay Area do this, and post the results to *ba.news.stats*. The SLAC reports were also archived for posterity, and this provided the raw material for studying the growth of news traffic. Comparison with selected reports from uunet [8] gives article counts of 85% to 90% for comparable periods, so this is representative of a typical large site that carries a full feed, but not every regional hierarchy carried by uunet.

Figure 1 shows the number of articles accepted per week by SLAC's netnews server. Note that this is *not* the number of articles received, as that number includes duplicates which have no effect on the amount of disk space required. Comparison of same-week data between two years suggests something close to a 67% annual growth rate. Aside from this rather stunning growth rate, the graph has a number of interesting features.

[3]The script to generate this report was written by Erik Fair and included in the B News distribution as `misc/report-awk`. The script was modified for C News, and C News itself was modified to provide some additional information in the log. This work was done by Larry Blair, with further modifications by Dave Rand. Karl Swartz made additional modifications and packaged everything up. This version is available via anonymous ftp from `ftp.slac.stanford.edu` in the file `pub/kls/news/c-news-report.tar.Z`.

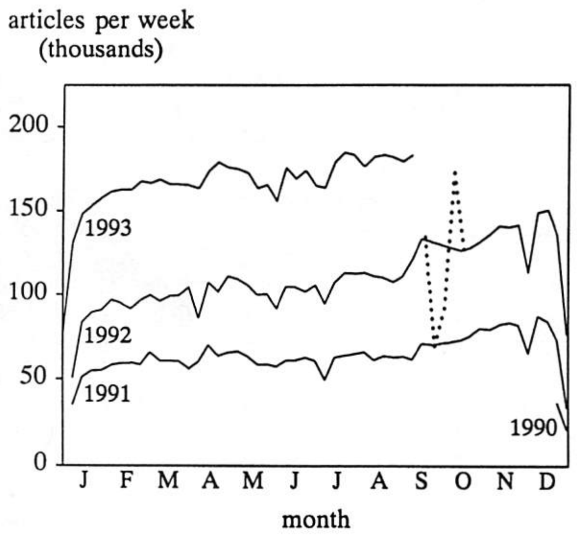

Figure 1: News articles accepted per week by SLAC's netnews server, with interpolation of data during the collapse of the old server and installation of the new one in September 1992

One noteworthy feature is the traditional September surge in traffic as many students return from summer vacations. The dotted line shortly after this onslaught in 1992 depicts the collapse of SLAC's old netnews server, along with the recovery after installation of the new server. (The analysis in this paper is based on the interpolated data for this period.)

Also interesting are the dips during holidays. Christmas/New Years was no surprise, nor was Thanksgiving, in late November. The dip in early July was a bit puzzling at first, but was quickly identified as the July 4th holiday – Independence Day in the United States. The size of the effect of a holiday unique to the U.S. did come as a bit of a surprise, given the worldwide nature of modern Usenet. (In retrospect, if this was surprising, Thanksgiving should have been as well.)

Other fluctuations appear to be correlated to certain annual events as well, but the evidence is weaker. Looking for many more patterns of this sort, while amusing, is probably no more productive than looking for patterns in clouds. Instead, the author turned to one of SLAC's physicists for some more scientific help.

A curve-fitting program (*Kaleidagraph*, running on a Macintosh), which uses the least fit squares fitting technique, was applied to the data. Several curves were identified that produced a good fit. Removal of five Christmas/New Years low points that were clearly not part of the overall trend fine-tuned the parameters of the curves a bit, producing the curves which are shown in Figure 2 along with their correlation coefficients (r). The equations for these two curves are

$$a = 49400 \times e^{0.0099w}$$

and

$$a = 49000 + 460w + 4.0w^2$$

where w is the number of the week relative to December 24, 1990.

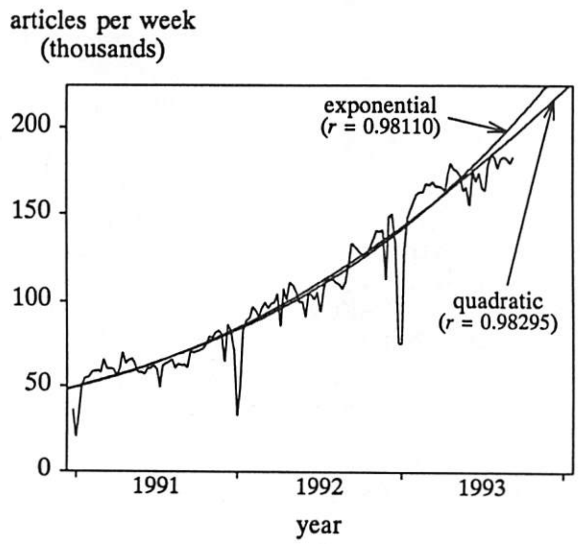

Figure 2: Exponential and quadratic curves fitted to actual number of articles per week minus five holiday low points

The adage that anything can be proven with statistics was reinforced when a least squares cubic fit was found that also seems to correlate well with the data. As seen in Figure 3, the long-rumored death of the net will occur on February 14, 1997 – no doubt a happy Valentine's Day for Usenet widows!

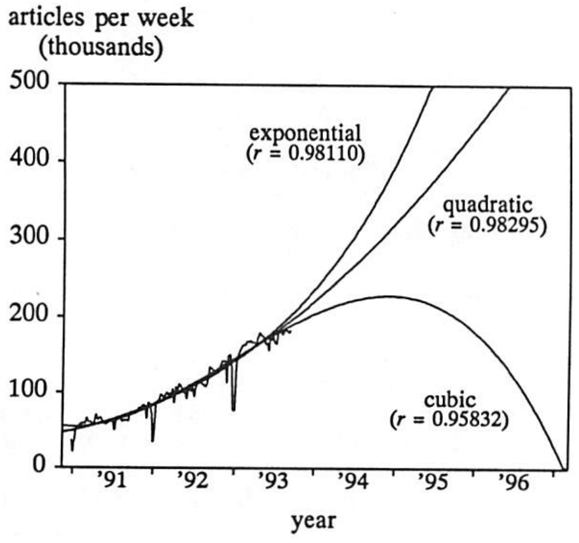

Figure 3: Death of the net predicted

There are several lessons here. One is that the sample is nowhere near large enough, and thus one should not depend on the resulting curves to accurately forecast growth too far in the future. Another is that good judgement and plain common sense needs to be applied to the interpretation of the results to avoid nonsensical predictions.

With this in mind, the exponential curve was selected for further use as being the closest to intuition. It's the most conservative choice in terms of disaster avoidance, as it forecasts the greatest disk needs. It is also remarkably close to the estimate of 67% annual growth, which a year ago had been estimated to be 65% and had predicted the past year's growth fairly accurately. In the interest of simplicity it was decided to just use the 67% growth equation

$$a = 50000 \times 1.67^{year-1991}$$

where fractional portions of a year are relative to January 1.

It's worth noting the rather large fluctuations in traffic levels. Because of this it isn't worth investing *too* much effort in precision when trying to forecast disk requirements – ballpark figures plus some extra space to accommodate peaks are sufficient to address the problem of knowing how much hardware will be necessary for the next year or so.

Size of Articles

The weekly reports generated by SLAC's netnews server don't include data on article size, and the only other local historical data was a single sample taken on March 30, 1992. There's little reason to expect much change, however, so data from several current sources were compared to establish a reasonable estimate of the current average size. This was then compared to the 1992 data, confirming the belief that article size is relatively stable over time.

First, `/usr/spool/news` on the SLAC server was scanned for all articles, ignoring non-article files such as those in the `out.going` directory, and being careful to note link counts to avoid multiply counting cross-posted articles. The result of this survey is shown in the first line of Figure 4. This result is biased because all groups aren't expired at the same rate. In particular, groups with very large articles such as those in the *alt.binaries.pictures* groups are expired much more quickly than other groups.

To compensate for this bias, several smaller scans were done of just the groups with non-default expiration periods, also shown in figure 4. All of these scans were done about twelve hours after the nightly *expire* run so the data represents a period equal to the expiration period plus one-half. Several SLAC hierarchies which are kept forever were ignored, since they don't apply to other sites. A weighted average was computed for the remaining data; this worked out to an average of 2,635.7 bytes per article.

For comparison, data from several sites were then harvested from *news.lists* for the August, 1993 timeframe. [8, 9] For the two weeks ending August 8, the average size of an article passing through uunet was 2641.5 bytes, while for the week ending August 7, the same metric for ira.uka.de was 2562.5 bytes, a difference of about 3%. For the six week period ending September 8, the average article size through uunet was 2586.2 bytes. These values agree fairly nicely with the numbers from the SLAC netnews server.

For the month of August, roughly the same period as the six week uunet data, [10] produced an average article size of only 2295.4 bytes. This is based on a survey of articles on disk, however, and thus is subject to the same expiration biases as the raw SLAC data, with which it compares quite closely.

The March 30, 1992 survey of SLAC's `/usr/spool/news` was done in the same manner as the current one, though no attempt was made to factor out expiration bias, since at that time all groups were being expired at nearly the same rate so expiration bias was minimal. The average article size at that time was 2,615.3 bytes, within the range of current sizes.

			article size		
newsgroups	inodes	references	total (KB)	average	expiration
all articles	424,617	497,392	948,238.8	2,286.8	—
slac,bes,hepnet	7,482	8,049	36,371.0	4,977.8	never
alt.binaries.pictures	144	156	5,743.4	40,841.8	0
other *alt.binaries*	128	134	1,428.1	11,424.8	7
control	2,665	2,665	1,279.9	491.8	3
all others	414,198	486,388	903,416.4	2,233.5	17
weighted (excluding *slac,bes,hepnet*)				2,635.7	

Figure 4: Survey of SLAC's article spool area: number of inodes and references to those inodes, total and average file size, and expiration periods for specific groups

To produce a reasonable working estimate for average article size, the average was taken of the current SLAC value (2,635.7) and the six-week uunet value (2,586.2). This gives an average article size of 2,611 bytes.

Effect of File System Block Size

Since each article is relatively small and is stored in a separate file, the block size (or fragment size, where relevant) of the file system is important. Figure 5 shows the article sizes from the SLAC netnews server along with the space needed given various blocking factors. Running netnews on a machine with a large block size and no fragments, such as an RS/6000 running AIX, will waste more than half of the disk! Fortunately, many systems use a variant of the BSD fast filesystem, but care must still be taken to configure the filesystem with 512 byte fragments to make the best use of the space. Little effort has been spent on determining this inflation rate with great precision – assuming 512 byte fragments and rounding up to 12% gives a fairly conservative estimate.

block	average	total (KB)	inflation
1	2,286.8	948,238.8	0.0%
512	2,543.4	1,054,663.5	11.2%
1024	2,786.2	1,155,322.5	21.8%
4096	4,866.6	2,018,018.1	112.8%

Figure 5: Effect of blocking factor on article storage

Directories, Inodes, etc.

Given the large number of files, several other filesystem features are also worth mentioning. With nearly 500,000 directory entries for the articles, directories themselves consume a non-trivial amount of disk space. Filenames of articles are simply sequence numbers and thus are normally quite short. Until a group hits 10,000 articles only 12 bytes will be required with a BSD-style directory. Cross-posts require additional directory entries, adding about 17% based on the numbers in Figure 4. Figuring 16 bytes per article for directory entries accommodates both cross-posts and very active newsgroups.

The inodes themselves also take up considerable space, but they are pre-reserved, so as long as one looks at usable space after *newfs* is done, inodes need not be considered. Since most articles are small, few if any indirect blocks will be needed.

The Bottom Line for Articles

Putting all of this together, the space required for the articles can be described as a fairly simple function of *time* and the *expiration* rate:

$$\begin{aligned} bytes &= 50000 \times 1.67^{year-1991} / 7 \\ &\times (2611 \times 1.12 + 16) \\ &\times (E_{articles}+1) \end{aligned}$$

One is added to the expiration rate assuming *expire* is run daily – an article arriving shortly after the time *expire* is run will get an extra day. Folding all the constants together we get

$$20{,}510 \text{ KB} \times 1.67^{year-1991} \times (E_{articles}+1)$$

To put all this into perspective, a site expiring news after ten days will need about 1.67 gigabytes of usable storage just for the articles by the end of 1994.

The History File

The history file is composed of three tab-separated fields, containing the article ID, times relevant to the expiration process, and a list of where the article appears. For example:

```
<42@hh.uucp> 752520600~- news.answers/42
```

An article ID is composed of a host-specific part and the hostname. B News used a simple sequence number as the host-specific part, but most other news systems use more complicated and thus longer values. Today, most news systems have been converted so the size of this portion of the article ID is probably fairly constant. Likewise, the conversion to domain names caused an increase in article ID length, but again this probably has little additional impact today. Thus, a simple survey of SLAC's history file was done to establish an average size for the article ID. Figure 6 shows the breakdown by size; the average over 823,810 entries was 33.9 bytes.

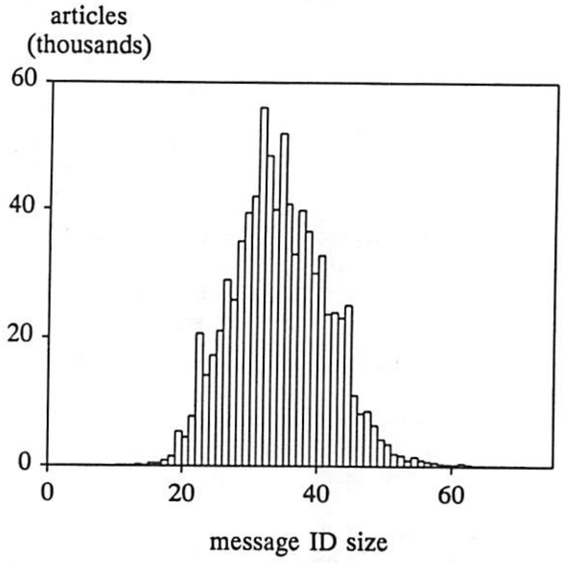

Figure 6: Article IDs by size (823,810 samples)

The second field is composed of two subfields separated by a tilde. The first part is the date and time the article was received by the local system in the normal UNIX form, i.e., the number of seconds since January 1, 1970 at midnight GMT. Normally, if the applicable expiration period is *n* days, *expire* will delete the article *n* days after it was received. If the article has an explicit expiration date, however, that date and time is stored in the second portion of the history entry's second field, and *expire* will try to use that as the expiration date for the article. (Explicit expirations may be overridden if they are too far in the future.) Few articles have explicit expiration dates, so this field of a history entry will usually be eleven bytes long, at least until September 9, 2001 (GMT).

The third field is a space-separated list enumerating the newsgroups in which the article appears, along with the article sequence number in each of those groups. This field and its leading tab will not be present if the article has expired. The number of entries with more than one subfield is just the cross-posting rate, which has already been estimated to be 17%. The size of each newsgroup/sequence number pair varies only slowly; it was determined by yet another survey of the SLAC history file to have an average size of 22.6 bytes.

The history expiration rate determines how long entries are kept in the history file. This is usually longer than the article expiration rate, so the oldest entries will lack the third field. If specified as being shorter than the article expiration rate, the article rate is used. (This case is not considered in the following calculations.) As with articles, one must be added to these rates to account for an extra day's worth of data, assuming *expire* is run once per day.

In addition to the history file itself there is also the index. Using *dbz* this is stored in `history.pag` (and `history.dir`, though that's tiny), and seems to stay around 10% of the size of the history file itself. Thus, at least for *dbz*, another 10% should be added to the size of the history file itself to allow space for this index.

Finally, when *expire* runs it creates a new history file and index, deleting the old ones only after installing the new ones. Thus the peak space requirement for the history file must account for two complete files and indexes.

Combining all of these factors, the overall space requirement for the history information can be stated as a function of the two expiration rates and time:

$$\begin{aligned} bytes = {} & 50000 \times 1.67^{year-1991} \,/\, 7 \\ & \times \left((E_{history}+1) \times (33.9 + 1 + 11) + (E_{articles}+1) \times (1 + 22.6) \times 1.17 \right) \\ & \times 1.10 \times 2 \end{aligned}$$

Log Files

The size of the log file depends upon various local configuration issues, including how many other sites a given article is fed to and how many duplicate articles are received. (With NNTP, duplicates should be zero or very nearly so.) A log at SLAC from a relatively busy mid-week day in early September contained 35,496 lines averaging 130.6 bytes long, a total of 4.4 megabytes. The number of lines can be expected to grow with the number of articles.

The logs compress quite well, often 3:1 or better, so if old logs are kept for a few days, compressing them will produce significant savings on space.

Overview and/or Thread Databases

Currently, SLAC is still using *trn* and its threading mechanism, which has used a bit less than 5% of the space required for articles, consistent with the author's projections. [6] This hasn't been studied in much detail since we plan on moving to *NOV*, for which the suggested 10% will be used as an initial estimate. [7]

Example

The following example demonstrates how these equations may be used to forecast the disk requirements for a netnews server through the end of 1994. The value for *year* is set to 1995, i.e., January 1, 1995. Articles in all newsgroups will be kept for 17 days, long enough for someone to go away for two weeks and not miss any articles, while history information will be kept for 30 days. The filesystems have a fragment size of 512 bytes. Article space is thus

$$\begin{aligned} & 50000 \times 1.67^4 \,/\, 7 \\ & \times (2611 \times 1.12 + 16) \\ & \times (17+1) \end{aligned}$$

or 2,804 megabytes. The space reserved by the BSD filesystem adds another 10% to this, bringing the total to 3,085 MB of usable disk space, well beyond the 2 GB limit of most BSD filesystems. The most expedient solution is probably a pair of 2 GB disks, using symlinks to split the articles over the two filesystems. This implies some effort at balancing newsgroups by volume, and hoping the symlink support really does work.

The history information will require

$$\begin{aligned} & 50000 \times 1.67^4 \,/\, 7 \\ & \times \left((30+1) \times (33.9 + 1 + 11) + (17+1) \times (1 + 22.6) \times 1.17 \right) \\ & \times 1.10 \times 2 \end{aligned}$$

or 224 megabytes.

This directory (`/usr/lib/news`) will also contain the logs. Assuming SLAC's log file is representative, and another 16 months will nearly double the size of the log, a daily log will need about 8.8 MB. Seven days of old logs, compressed to about one-third of their original size, adds another 20.5 MB. Adding a few megabytes for other files and the 10% reserved by the filesystem means a partition of about 300 MB is needed.

The *NOV* system will be used for threading; the 10% estimate implies 280 MB for this purpose. Leaving some room for error, and the filesystem's 10%, a 400 MB partition would probably be prudent.

Partial Feeds and Other Adjustments

The equations derived in this paper are based on carrying a full Usenet feed along with a number of other common hierarchies, e.g. *alt* and *gnu*. A uniform expiration rate across all newsgroups is also assumed. If these conditions are not true, some tuning will need to be done. A good source for raw data is the collection of monthly Usenet reports posted by Brian Reid, for example see [10].

Sites carrying a partial feed can identify major groups or hierarchies which they do not carry and, using the monthly Usenet reports, determine what fraction of the total articles are being carried, multiplying the computed article rate by this fraction. Depending on the groups, the average article size may also require some tuning – the data for this is also in the monthly reports.

Tuning for varying expiration rates can also be done using these monthly reports. This is essentially the reverse of the weighting done in Figure 4.

Conclusions

Historical data on system activity is quite valuable, as it can be analyzed to assist in planning for future system growth. One must be wary of trying to read too much into statistical results, however, especially on rather small samples. Common sense is needed to recognize unreasonable results.

With regard to Usenet, growth of the net can be predicted reasonably well, at least over the span of a year or two. With some analysis of how this growth influences the growth of various databases, disk requirements for a Usenet server can be forecast sufficiently to permit acquisition of new hardware or adjustment of administrative controls before problems arise.

References

1. Tim O'Reilly and Grace Todino, *Managing uucp and Usenet, 10th ed.*, O'Reilly & Associates, Inc., Sebastopol, California, 1992.
2. Anatole Olczak, *Netnews Reference Manual, 2nd ed.*, ASP, Inc., San Jose, California, 1989.
3. Geoff Collyer and Henry Spencer, *Installing and Operating "C News" Network News Software*, February 1993.
4. Tom Limoncelli, ed., "INN FAQ Part 1/3: General Information, how to compile, how to operate," *news.software.nntp (Usenet newsgroup)*, August 12, 1993.
5. Wayne Davison, mthreads(8) man page for trn 2.2.
6. Wayne Davison, README for trn 2.2.
7. Rob Robertson, ed., "FAQ: Overview database/NOV General Information," *news.software.readers (Usenet newsgroup)*, September 8, 1993.
8. "Total traffic through uunet for the last 2 weeks," *news.lists (Usenet newsgroup)*, bi-weekly report.
9. Matthias Urlichs, "NewsStats, article counts," *news.lists (Usenet newsgroup)*, August 7, 1993.
10. Brian Reid, "USENET READERSHIP SUMMARY REPORT FOR AUG 93," *news.lists (Usenet newsgroup)*, September 9, 1993.

Acknowledgements

Special thanks to Tom Glanzman for doing the curve fitting and for teaching me a bit about statistics. Other people who helped in various ways include Mark Barnett, George Berg, Chuck Boeheim, Les Cottrell, Krissie Griffiths, Don Pelton, Mike Sullenberger, Bebo White, Lois White, and others whose contribution is not diminished by my failure to remember them. My gratitude goes to all of them. Thanks, too, to Alexander for his patience – and for *not* finding another skunk.

Author Information

Karl Swartz is a member of the Server Systems Group within SLAC Computing Services at the Stanford Linear Accelerator Center, where he is the resident UNIX guru. Prior to joining SLAC, he worked at the Los Alamos National Laboratory on computer security and nuclear materials accounting, and in Pittsburgh at Formtek, a start-up that is now a subsidiary of Lockheed, on vector and raster CAD systems. He attended the University of Oregon where he studied computer science and economics. Karl instructs at high-speed driving schools and enjoys good food and good beer (though not while driving) and hikes on the beach with his Golden Retriever, Alexander. Reach him electronically at kls@chicago.com or kls@unixhub.slac.stanford.edu.

Simplifying System Administration Tasks: The UAMS Approach

Roland J. Stolfa – Oklahoma State University

ABSTRACT

The User Account Management System (UAMS) is an extension of the original User DataBase (UDB) system presented at the USENIX Large Installation System Administration Conference in 1990. This paper describes the extensions of the UDB system from a single administrative entity tool for a distributed set of computers to a multidepartmental system over the period of three years since the first paper. It also covers the added features that have been developed, such as support for Novell networks and POP clients.

Introduction

In the university environment, a computer system administrator's job can be quite diverse. For instance, here at Oklahoma State University in the Computer Science Department, there have been times when two system administrators have been called upon to manage over 40 hosts, each with a separate password file, backup requirement, and operating system type. In addition, these same two individuals were required to field questions from over eight hundred users on these 40 hosts as well as the other hosts run all over campus.

As was made clear very early on, this situation had to be improved upon. The first and foremost area of concern was in improving the generation of accounts. At first, user accounts were stream lined to be a simple prefix, followed by a set of digits. An example would be "fs" for "file structures" followed by the numbers one through the number of students in the class. One problem with this was that we found several students having several accounts on the same system for purposes of one semester's class work. Further, pinning down exactly which student was associated with which account was not a very easy task: a task made more important with the attachment of the OSU campus to the Internet.

In short, there was no real solution to the user account creation, deletion, and management problems for a university. This situation lead to the original development of UDB, The User DataBase System, in 1987. Further refinements lead to the presentation on UDB at the Large Installation Systems Administration Conference in 1990 (available as OSU-CS-TR-90-04)[1].

After the first paper on UDB, several other colleges and departments within OSU decided to participate in the types of services UDB was providing. However, to extend UDB to that domain would have required all departments that wanted to participate share one database on one host. For various practical reasons, a distributed solution had to be found. The OSU Computer Science Department found itself in need of a system of providing database services to a majority of the campus. A system was needed that could maintain a campus wide flat name space, for both logins, or Universal Computing Identifiers (UCI) as we call them, and numeric user ids (NUID), allowing separate administrative entities access to only those pieces of information that directly relate to their organization. Due to the development of UDB and the extensive number of hours spent developing the code, a conscious effort was put forth to extend UDB to meet the new challenges. The remainder of this paper discusses some of the extensions to support this effort. Several other account maintenance systems were presented at the 1990 LISA conference. Some of their high points are summarized below.

In ACMAINT[2], a central database was presented to allow a single system administrator to manage computer account creation across a heterogeneous set of computers. However, it utilizes daemon programs running on both the server and the clients that rely on TCP/IP networking facilities. Although these facilities are gaining popularity here at Oklahoma State University, not all hosts have TCP/IP, hence this approach was not usable here.

In GAUD[3], a central database is accessed by many hosts over Remote Procedure Call (RPC) protocol to allow access by the various offices that might allow or deny access to a particular user to a particular machine. However, GAUD suffers from the reliance on the source code to the operating system, an item not all universities have. Furthermore, here at OSU, RPC is not available on all hosts, hence its unsuitability.

In newu[4], a functionality that is already in UDB was described (i.e., the ability to create and delete accounts on a foreign host). It suffers from the same problem UDB had, in that it only worked within one administrative entity.

In Uniqname[5], a system of merging existing accounts with a 'global view' is presented. As UDB started with a unified 'global view', Uniqname offered a solution to something that was not a problem in our case. In addition, it presents a solution to a problem that UDB did not even attempt to address, that of preferred mail box address.

Data Analysis

An analysis of the data involved in account creation was undertaken to determine the easiest way to modify UDB to address the issues raised while trying to support the majority of the campus. This left an understanding of which fields of the database needed to be 'global' and which ones could be 'local' to the administrative UAMS site.

Global Data Fields

Most of the global data fields are overwritten in the process of receiving a new data feed from the registrar. The UCI and NUID are the exceptions in that they are generated only the first time the given user's record is entered into the system.

Student/Faculty identification number (ID)
Student/Faculty ID card issue number (ISSUE)
Student/Faculty full name (FULLNAME)
Universal Computing Identifier (UCI)
Preferred Numeric User ID for NFS (NUID)
Student department affiliation (MAJOR)
Automatic rights (AUTO)

Figure 1: Global Fields of the Database

The data analysis concluded that all slave UAMS sites across campus would have to share some part of the the global database maintained on the master site. The slave UAMS sites would treat this data as read only, allowing the master UAMS site to overwrite these fields at will. The global fields are treated read only on the master UAMS site after the initial creation of the users record. This is because the entire list of UCIs and NUIDs are kept unique on the master UAMS site. Once generated uniquely on the master site, these global data fields can be transmitted to any of the slave UAMS sites while still guaranteeing the data integrity (i.e., no duplicate UCIs or NUIDs).

Local Data Fields

The local data fields are unique to each administrative UAMS site. This allows each UAMS site to have control over the special case users without infringing on any other UAMS site.

Clear text initial password of user (PASSWD)
Encrypted initial password of user (EPASSWD)
Manually granted rights (GRANTED)
Comment field (COMMENT)
Last update time of this record (LUPDATE)

Figure 2: Departmental Fields of the Database

One consequence of this splitting of each record is that each administration would be able to set the default initial password, while maintaining the same UCI. This would help maintain some level of security between UDB hosts. This arrangement disables one user, knowing their UCI on one UDB administered host, from logging in ad-hoc to other UDB administered hosts based purely on the knowledge of the original password. It also prevents an individual user's account information from being of much use to someone else, as the initial password would be different between UDB administrations.

Each administrative UAMS site would also have a separate and unique GRANTED right field for each user in their system. This allows each site to specify unique (and possibly conflicting) URIs, or Universal Rights Identifiers as we call them, to give access to different clients (hosts, etc.). On the master UAMS site, the GRANTED right field is also used to select those special case users, like 'root', that need to go to a slave UAMS site in addition to those users destined to go to the slave site because of enrollment information. However, as the GRANTED right field on the master UAMS site does not go with the record to the slave site, the slave UAMS site never sees that URI.

Another result of this data analysis is that each department is allowed their own comment field (COMMENT) for each user. That way, any comments on a user are held in the confidence of the commenting department.

Transport Methods Analysis

Some method had to be found to get the parts of the database distributed amongst the various UDB sites. As previously mentioned, RPC could not be used because some hosts did not have it. Some hosts, although fewer than in the past, lacked TCP/IP, so things like a socket based transfer protocol were out. This left the original UDB's solution of lowest common denominator, email.

Although email has served UDB quite well for a number of years, it suffers the same security risks as any other information exchange media. With a good understanding of this, we were forced into using it to communicate with these lowest common denominator systems. We have attempted to make it as secure as possible, but more from a data integrity stand point. This was viewed as very important in assuring that the data destined for a particular site is the correct data for that site.

So for our installation, the original choice was once again validated. The transport mechanism for sharing the data between the various UAMS sites would be email.

Sharing the Data

In order to facilitate the sharing of data, a simple master/slave model was chosen. This represents the administrative association of the 'global fields' of the UAMS databases amongst each other. Hence there is a master UAMS site. This is the site that receives the enrollment data from the registrar. It is also the only UAMS site that can definitively assign UCIs and NUIDs. Whenever any slave UAMS site needs a UCI and NUID for a user new to it, it must defer to the master UAMS site for the definitive information. The master UAMS site also maintains the master copy of the enrollment data for all students enrolled in classes that all of participating UAMS departments have authorized the master UAMS site access to (i.e., all the classes that the participating departments teach).

The master/slave model allows the master UAMS site to select all of the records for a slave UAMS easily. This is typically based on enrollment data held in the AUTO and MAJOR records (for students). Additionally, to select all non-standard records (for such things as 'root', 'uucp', faculty, etc.), a specialized GRANTED right, unique to that slave UAMS departmental administration, is used.

To provide each participating department autonomy over their users, only the 'global fields' are shared between the different UAMS sites. This implies that any URI given to a user on the master system as a GRANTED right is not propagated to any other UAMS slave site. This includes the specialized GRANTED right, as the GRANTED field is not in the 'global fields' list of data being shared. This also allows each departmental UAMS to have overlapping (and possibly duplicated) GRANTED right URIs. Further, it allows each departmental UAMS to use the COMMENT field as they see fit, without having to conform to some standardization scheme.

The Server/Client Model

A server/client relationship exists between any UAMS site and a system that is administered by the owner of the UAMS server. In this system, any server, either a master UAMS or a slave UAMS, may provide a data feed to a client system. The format of the data delivered to the client is client specific, and is typically used to generate some end product specific to that client. As way of example, the rest of this section will discuss the generic UNIX client system.

```
...
uci:epasswd:fullname:granted:major:auto:nuid
...
```

Figure 4: Sample datafeed for UNIX client system

In this model, all URIs are propagated to all client systems (along with the information needed to create /etc/passwd and /etc/group) from their administering UAMS server to allow the generation of the URI database for each host. This allows each departmental machine to make use of all of the enrollment data, the major code, and all of that departmental UAMS server's unique URIs for its own ends. One of the uses of the URIs that has been implemented here at OSU is the ability to run various programs (similar to access control lists). Another is the automated maintenance of mailing lists.

Final Design Criteria

There have been several ideas presented so far that guided the design process of UAMS. Among the most important were:

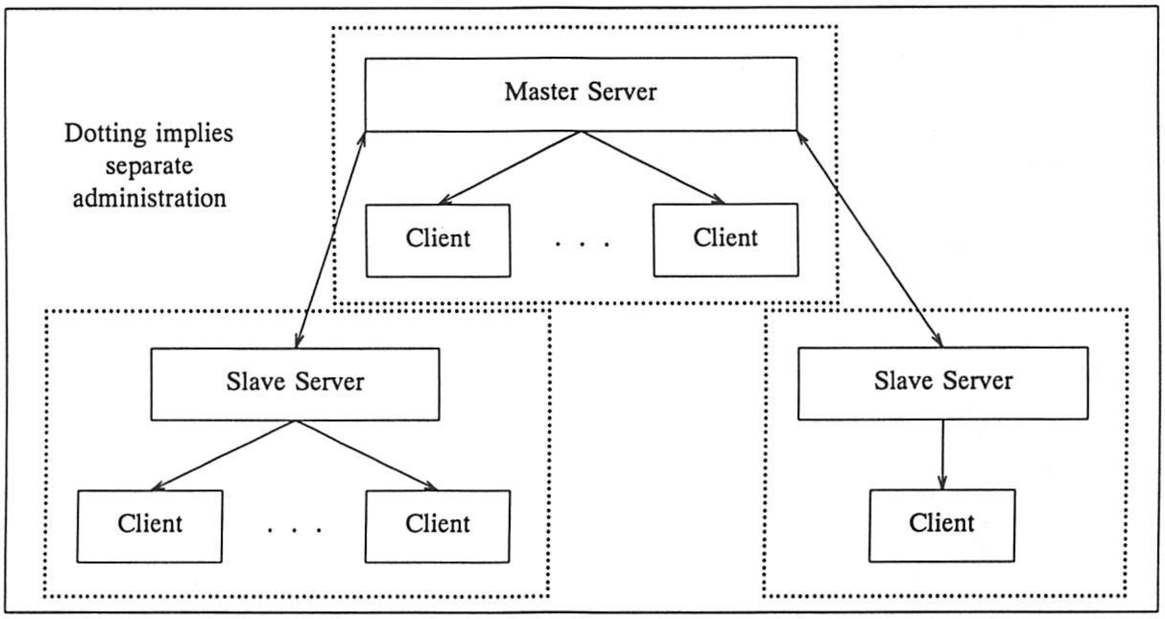

Figure 3: Master/Slave and Server/Client Relationships

- Centralized master copy of data base for the main unchanging parts of the database.
- The entire package should work with mail. No special use of or dependence on TCP/IP or RPC was allowed because not all hosts on campus have such capabilities.
- The system should, as did UDB before it, work with all the UNIX utilities without having to change said same utilities. This is because we do not have source code for all of the hosts on campus.
- There should be no long-running daemons in the system.
- The system should not use any special operating system specific code, nor any commercial product (such as a commercial database package). This is because for the vast number of platforms that it would have to support, the inherent cost would be prohibitive.
- As we want to distribute the resulting system, no AT&T derived source code was to be included in the system.

The UAMS/UDB System

The User Account Management System (UAMS) is an extension of the original UDB system to encompass all of the changes discussed in the previous sections while adding several additional new capabilities. Some of the modifications are listed below:

The Master to Slave Data Feed

The original UDB used a selection list, called the Rights Access File (RAF), to select which URIs (a combination of AUTO, MAJOR, and GRANTED field's contents) granted you access onto a particular system. A simple extension of this was used in UAMS to select out those rights that granted a record passage to a slave UAMS site. As it turned out, the only additional functionality was a simple wildcarding facility so that such choices as 'MATH*' (to select all classes taught under the 'MATH' heading) could be made. These selected records were then sent through a filter and mailed to the destination UDB.

```
...
id:issue:fullname:auto:major:uci:nuid
...
```

Figure 5: Sample master to slave data feed

Special Case Users

To handle the special case users, such as 'root', 'uucp', et.al., several things had to be overcome. First, these users did not have OSUIDs, any enrollment data, and quite seldom a FULLNAME. To cover these cases, as well as the case of the occasional guest account (or odd software package) that did not have an OSUID, a simple hueristic was formed of taking the proposed UCI (say 'root') and using a '+' prepended to the UCI as the OSUID. Thus 'root' would have the OSUID of '+root'.

This simplifies some areas of system administration. For instance, all of the users with plus-records ('+root' would be a plus-record) have no OSU student or staff id. Therefore they need not be selected for loading into our card key lock software[1]. Also, they are typically what we call 'mechanical accounts', i.e., they come with an operating system and do not have a physical person behind them. So when we scan the password file for accounts to set no-login, we can use UAMS as an aid in this process (this is not an enforced function of UAMS, merely an example of using the data UAMS maintains).

Anti-Rights

As in most any other university setting, students will be students. As UAMS was applied to an ever larger body of students, it was inevitable that a student would need to be kicked off of a machine due to an infraction of the rules. Shortly before this became necessary here at OSU, I had developed the anti-right. With this granted right, a user could be excluded from a machine, regardless of their other rights.

Let's say the user 'foo' is to be kicked off of the machine 'A'. However, currently 'foo' is granted access to that machine due to either a class enrollment right (in an AUTO field in this user's record), or their MAJOR (if they are a guest on that machine, indicated by a URI in this user's GRANTED field of, say, 'A', the granted right 'A' is simply removed). To delete all those rights would be unreasonable. First off, the next time UAMS received this user's enrollment data from the central university data base, the AUTO field enrollment right would return. At this time, the MAJOR code would be restored also. As these are not solutions, the anti-right was developed. In the case of 'foo', a GRANTED anti-right would be given as '!A'. In fact, if we just wanted to lock 'foo' out for a specified period of time, an expiration date could be added, giving an expiring anti-right of '!A-YYYYMMDD'.

These anti-rights do nothing abnormal to the record of the user. Instead, they alter the list of users selected to go to a site, 'A' in this case, to exclude this user. If this same user has some other granted rights, for instance this user has an account on the same department's POP server, their POP server rights are unaffected.

Comment Fields

As with any major software project, oversights are pointed out as soon as the code is delivered.

[1]As described in [1], the Computer Science Department runs a card key lock system on several labs within the department.

One of the other departmental UAMS administrators found a good use for a comment field, but discovered my oversight in not having one. So I enhanced UAMS to have a per user comment field.

This field is not automatically filled in, or in fact created, for every user. Therefore it was implemented as a sparse record within the database, similar to the AUTO and GRANTED fields records within the database.

New Novell client

The Novell version 3.11 client came about because of one of the other departmental UAMS administrators. Within the other department, Novell was used to link together several personal computers within a student lab. However, all of these computers had to be configured for users, with much the same information, just as the UNIX hosts that UAMS already served. As this is just another type of host, using unique login names and passwords, a new client was written to provide the information.

After researching the Novell manuals and considering the options, the client was written to generate a data file for the Novell user administration program 'MAKEUSER'. The UNIX side would keep a list of the users currently authorized for a Novell site, compare that with what UAMS was giving it, and generate add and delete commands as necessary. This preserved the feel of the UNIX client, without having to write a program for a personal computer.

New POP Client

The Post Office Protocol (POP) client came about because there was an interest within several departments to provide mail to personal computers. The POP system, as distributed with the RAND Corp. MH mailer, was chosen by some of them to fill this need. Again, this system had to be configured for users, just as the UNIX hosts did. As POP uses a password file that is in many ways similar to the UNIX password file, this client required several minor changes to one of the existing clients, and it was up and running.

New Administration Interface

After some time of using UDB, I found myself facing a problem. UDB had no real command line interface mechanism. This proved to be a hassle when I wanted to change the name of a particular URI to all those users who were granted it. In addition, I was trying to lure some DEC VAX/VMS system administrators at the time to use UDB. As a result I wrote a simple command line interface for UAMS, similar to DEC VAX/VMS AUTHORIZE. It operates on a single field of a single user's record at a time strictly from the command line.

Benefits

The UAMS package here at OSU has been operational for about three years now, with UDB operating for about three years before that. The system currently supports over 15,000 user account records across three colleges. They are split approximately as follows (the remainder are holdovers between semesters):

- 5500 on a collection of Silicon Graphics, Sun Sparc, Sun 3/60, and AT&T PC in the College of Architecture, Engineering, and Technology.
- 1200 on 26 IBM RS/6000 in the College of Architecture, Engineering, and Technology.
- 140 users on seven Sun Microsystems computers in the Department of Agricultural Engineering.
- 150 users on 12 Sun Microsystems computers and 3000 users of a PC/Novell network in the Mathematics Department.
- 1200 users on a Sequent Symmetry S-81 computer in the Computer Science Department and 700 users of a card key lock system.

Listed below are some of the benefits we have seen as a result of UAMS.

Uniqueness

When a user first enters the UAMS system, they are given a unique login. This allows such things as backups and mail to be uniquely identified across campus with a user. Since this information is keyed to their Oklahoma State University id card, even years hence, this student will be able to be uniquely identified and their files retrieved with a good level of certainty that they are being retrieved for the correct person.

Start of Semester Crush

At the start of each semester, there is a large enrollment influx of new users, mostly students. To create all of these accounts by hand would be impossible. With UAMS, to add an entire class of students to a machine is as simple as adding three lines to a configuration file, running a new enrollment database through the system, and installing the resulting password file. This entire process can take as little as 30 minutes.

Mail Lists

In the past, different instructors have tried to keep track of which students are in their class for purposes of a class mail list. UAMS automates this procedure and makes sure that the list is correct, up to the last enrollment data feed.

Interdepartmental Data Sharing

In the past if a user wanted to have the same data in two accounts, their only real options were along the lines of mailing the data between the two accounts. With UAMS in place on all of the hosts involved, it is possible now to automount the data across campus. This is a direct result of having the UCI and NUID the same for each user on any UAMS administered host.

Conclusion

UAMS offers system administrators in a distributed departmental environment a unified environment for the administration of users. It does so without infringing on the local department's internal organization while offering support for quite a number of different clients (UNIX, Novell, POP, etc.). UAMS is quite extensible to any environment where a simple UCI, password, and or numeric user id is needed.

Credits

I would like to thank Mark Vasoll (Computer Science Department) for helping proof read this paper and for giving me inspiration on various sticky bits. I would also like to thank Rod McAbee (College of Engineering, Architecture and Technology), Russ Smith (Mathematics Department), and Clay Couger (Agricultural Engineering Department) for their interested support.

Author Information

Roland Stolfa is a Software Specialist on the staff of the Computer Science Department of Oklahoma State University. His interests include network simulation, robotics, aeronautics, and system administration automation. Mr. Stolfa has worked for the university since 1986. Reach him via electronic mail at rjs@a.cs.okstate.edu. His U.S. mail address is Computer Science Department; 219 Mathematical Sciences Building; Oklahoma State University; Stillwater, OK 74078.

References

[1] R. Stolfa and M. Vasoll, "UDB – User Data Base System," in *USENIX Conference Proceedings, Large Installation Systems Administration IV*, Colorado Springs, October, 1990, pp. 11-15.

[2] D. Curry, S. Kimery, K. De La Croix, and J. Schwab, "ACMAINT: An Account Creation and Maintenance System for Distributed UNIX Systems," in *USENIX Conference Proceedings, Large Installation Systems Administration IV*, Colorado Springs, October, 1990, pp. 1-9.

[3] M. Urban, "GAUD: RAND's Group and User Database" in *USENIX Conference Proceedings, Large Installation Systems Administration IV*, Colorado Springs, October, 1990, pp. 17-22.

[4] S. Schaefer and S. Vemulakonda, "newu: Multi-host User Setup," in *USENIX Conference Proceedings, Large Installation Systems Administration IV*, Colorado Springs, October, 1990, pp. 23-26.

[5] W. Doster, Y. Leong, and S. Mattson, "Uniqname Overview," in *USENIX Conference Proceedings, Large Installation Systems Administration IV*, Colorado Springs, October, 1990, pp. 27-35.

System Administration as a User Interface: An Extended Metaphor

Wilson H. Bent, Jr. – University of Southern California

ABSTRACT

Many people have spent many hours searching for the right user interface. Many people have spent many hours searching for the definition of the right user interface. Many people have spent many hours decrying the wrong user interface (whatever that may be). One user interface which is often overlooked is the System Administrator.

What are the qualities of a good user interface? Some of the features most often mentioned are:

- User-friendly
- Extensible, adaptable to the user's skill level
- Gives useful error messages
- Hides underlying system's gore
- Offers easy access to helpful information
- A terrific look-and-feel

This paper will discuss each of these features with regard to the interface between the user and the system administrator.

Introduction

What is a user interface, generically? It's a layer that sits between a person (a "user") and whatever resource that person is trying to use – information, data manipulation, or access to a network. It's the way a user controls a resource. A toaster has a fairly simple user interface: a knob to set the amount of toasting, and an on-off slider. Notice that, even for a job as simple as toasting bread, it takes two controls to provide the flexibility needed.

Television sets used to get by with two controls: the channel selector and the on-off switch, which doubled as a volume control. Such technical details as the brightness and horizontal hold knobs were hidden away, and usually only needed adjusting once. As TVs have become more advanced, more flexible, and more "sophisticated," the user interface has become more complicated, so much so that even a basic model TV comes with a remote control with a couple dozen buttons.

Because of its capability of being programmed to perform a wide range of tasks, the level of complexity in a computer can be much higher than any other device. As a result, there is the possibility of a wide range of user interfaces: a spreadsheet performs a task which is very different from a word processor, and both are different from a flight simulator. However, their user interfaces are not entirely dissimilar. Programmers and system designers have, over the years, defined qualities of a good user interface. These qualities allow the users to get their jobs done with relative ease, without feeling that the computer is "getting in the way."

We're used to thinking of the term "user interface" in very narrow terms; specifically as what appears on a user's screen and how the user interacts with the computer. However, there's more to using a computer than that. Other interfaces which can make a computer easier or harder to use include the ergonomics of the workspace – and even the manuals.

As systems administrators, we ourselves are another user interface. Our function is to make resources available to users, and we can learn some lessons from the designers of graphical user interfaces.

User-Friendly

No doubt many systems administrators see the term "user friendly" with regard to their users and groan "What, *me* be friendly with *them*?" But remember, it doesn't have to be a case of us-*vs*-them, it can also be a case of client and customers. It can even be a case of peers: two people who are experts in their respective fields, one of which happens to be the field of systems administration. That's a nice, pretty-sounding side of the coin, but here's the flip side: chances are that if you are not user-friendly, it'll get back to your boss, which means that it'll get back to you.

Your attitude makes a difference. Nordstrom's department store is doing well and even expanding despite having higher prices in a time of tight budgets. Why? Because they have a widely-known policy that the customer is always right, and their staff is trained to act on it. Customers are willing to pay more, knowing that they'll get treated with respect.

There are those who argue that caution should be used when it comes to user-friendliness, and that "the customer is always right" is not always the right motto. They warn that if you promise users the world, they'll be sure to ask for it, and more. If you work too hard at solving users' problems, they will learn to see you as someone who will solve *all* their problems.

Such caution is a worthwhile thing, but be sure you've reached the point where it's appropriate to be cautious. Make sure that you've shown the users that you are capable of solving *some* of their problems before you advise them that you can't or won't solve *all* their problems. If the customers are not always right, be sure they know why they're not, and be sure they know when they *are* right. Remember, a diplomat is someone who can tell you to go to hell, and make you look forward to the trip.

You may be solving every user's problem with speed and expertise, but if you are doing it with an attitude which implies "I'm smart and you're not," you will only make the user go away not wanting to return. This may at first seem like the desired result, but in the long run it means you'll run out of users and thus put yourself out of work. The users will still exist and still have questions, they'll just find someone else to answer them.

Also, if you establish a friendly working relationship with the users, you can expect (or even ask for!) help from them at times when you need it. When you develop a good relationship with users who aren't afraid to talk to you, you're more likely to get early reports of broken stuff, and a chance to fix it before major damage is done.

Extensible, Adaptable to the User's Skill Level

No advanced user likes an interface which requires twelve button presses to delete one file. Some disk formatting programs strike me as a bit extreme: "Are you sure you want to do this (Y/N)? Are you REALLY sure (Y/N)? Press Q to quit, any other key to stop."

Similarly, the reason that GUIs are becoming so popular is that there are oodles of users at the other end of the spectrum who don't want to learn either `ls` *or* `dir`, who can't remember `del` *or* `rm`, and who can cut and paste a lot faster than they can write and read a temporary file.

It's also easy to find examples of manuals at both ends of the spectrum: many DOS programs come with a slim brochure which spends more time showing you how to insert a disk into a drive than how to use the program. Conversely, the high-end, high-price programs always seem to come with many shelf-feet of documentation but no hints on how to locate the one bit of information you need.

If a novice user asks you how to find a missing GIF file and your answer is a simple "`find . -name '*.gif' -print`", that user is not going to go away happy. Some users will benefit from a minute or two spent explaining the options to `find`, others will be better suited if you install an interactive shell script which results in running `find`. A more experienced user, however, may only need a brief reminder of some of the less-frequently-used options to `find`.

The thing to remember is that these are all valid users. They have a job to get done, and depend on you, the administrator, to help them, advise them, and set up a system for them which lets them get their job done with a minimum of fuss. Your response to their questions should match their skill level, so that you are neither insulting them by being overly simplistic, nor swamping them with your wizardly knowledge.

Journals such as the *Communications of the ACM* often publish papers on help systems and user interfaces which make use of artificial intelligence and a natural-language interface to access information. What they're trying to do is what you already do – except your interface is based on *natural* intelligence, and chances are you have a greater vocabulary than their natural-language interpreter! By engaging in an interactive dialog, an administrator can quickly focus on a user's needs and wishes, and respond in terms the user can understand. Being human has its advantages.

Some people want to learn how things work – they like to find out what makes things tick. Others are completely learning-hostile, and just want you to make them a magic bullet. Both kinds pose different challenges. The secret of user-friendliness and extensibility is: adapt. Learn to figure out the skill level of a user as quickly as possible.

A Diversion: The Systems Administrator as a Meta-Interface[1]

The real advantage of the human interface you offer to users is that you can go beyond just answering their questions. You can explore with the user such questions as "What is it that you're trying to do?" or even "Why are you trying to do that?"

If a user asks you to restore some or all of her mail file every week or so, you can solve her individual problem each time she asks. Or, you can go one step further and create a method for her to restore her own mail files, which empowers her and reduces your workload – apparently a good solution for both of you. But to take it one step further, you can explore with her why it is that she keeps having trouble with her mail: are the commands for "save" and "delete" similar, or are those two buttons in her graphical mail interface too close for comfort? Has

[1]Every serious paper should have at least one *meta*-discussion.

she uncovered a bug in the mail program? Is she receiving large mail messages which are filling her file system?

As a human interface, you can be more than just a help button.

Gives Useful Error Messages

Probably every paper on user-friendly programs ever presented at a USENIX conference has mentioned `ed`'s "?". Although `ed` is still cryptic, newer programs are sometimes quite helpful when reporting errors, although not so many that we can stop writing papers about the true-yet-unhelpful "not a typewriter" message.

Probably every paper on working with users ever presented at a LISA conference has pointed out that saying RTFM is not a good thing. Unfortunately, RTFM still gets said time and time again, so I have to speak up for the voice of reason: how about suggesting WHICH manual page to read? How about even mentioning which BOOK to read? As a Sun user, I highly recommend the AnswerBook software. Despite the fact that searching for a topic usually results in too much and occasionally too little information, it remains an easy-to-use interface to the manuals.

It's worth the time to make sure that users are informed when something doesn't work, regardless of the type of interface. If you're writing a shell script, it's more helpful to catch signals and be prepared for possible failures than to have the users confronted with core dumps. If you're referring them to another person or group, it's more helpful to explain why that group would better serve their needs than to just pass the buck. If you're helping them as they navigate a directory tree, it's more helpful to briefly explain access permissions than to just say "You can't go there." If you're on the phone with the user, it should be an absolute requirement that you say "see dee user local..." at a pace the user can understand, and when you slow down, that you keep that pained tone out of your voice!

Hides Underlying System's Gore

The joy of using computers is that they can do tasks that you or I can do, but they do them faster and without error.[2] In performing these tasks, a computer goes through many steps, but a good user interface hides that fact from the user.

The ability to cut and paste is considered essential to a GUI, but the underlying mechanics can be fairly difficult. Drag-and-drop looks equally easy from the user's standpoint, but is still not well-defined after five releases of the X Window system.

[2]That's the theory, anyway.

Even a simpler user interface such as the Bourne shell or the DOS command interpreter hides a large amount of path searching, `fork()`s, `exec()`s and `wait()`s. Under the version of UNIX I use, an 'easy' command such as `date` calls over 20 system calls and library functions, and of course the library functions themselves make other system calls...

What gore are you hiding as an administrator? Not just the internals of the UNIX system, but the internals of your administration system. When users ask for files to be restored, they're asking to be insulated from the gore of your backup scheme and tape library. If your tape library doesn't seem like gore to most of your users, either it's too small or they're as anal-retentive as you.

Your job as an interface to a fairly complicated system is to provide a method by which users may get a simple job done ("restore this file" or "print this memo") without needing to know the details. If they WANT to know the details, you should be willing and able to provide them, but even the experts usually just want to get the job done.

A Diversion into Paradigm Shifts[3]

A sure-fire way to start a conversation (or worse) among computer users is to say "The Macintosh user interface is more intuitive than MS Windows." Things will quickly degrade into comparisons of numbers of buttons on a mouse *vs* number of clicks needed to perform a task.

Yet, watch a user go from one GUI to another. How hard is it to adapt to a different window-frame decoration? How quickly can you figure out how to resize a window? Despite the frustrations, there is enough general similarity among GUIs to at least know where to start.

Administrators of systems from Sun Microsystems are discovering that a sure-fire way to start a conversation (or worse) among their peers is to say "I just switched to Solaris, and what a pain!" Things will quickly degrade into comparisons of `a.out` formats and options to the `ps` command.

Okay, so what's the point? Just as you, the system administrator, encounter problems in dealing with change in, *e.g.*, operating systems, so do the users legitimately encounter problems with changes in other, "simpler" (to you) interfaces or systems. Just because you consider those systems to be "easy to understand" or simple, doesn't mean that the user community will feel the same way. Any changes you introduce are going to require an adjustment

[3]Every serious paper should use the term *paradigm* at least once.

period for the users. Preparing them for the change will make life easier for everyone.

Offers Easy Access to Helpful Information

Another strong point of GUIs is that they often make it easy for the user to get help for a specific item or area. Two examples of this are Sun's Help system under OpenWindows, and Apple's balloon help under System 7. Such systems allow the user quick access to information about a limited topic, without having to wade through a stack of manuals.

For systems administrators, being able to provide such a service to users will not only make life easier for them, but easier for you, too. I have seen quick tutorials for both `emacs` and `vi`; having either or both on-hand, ready to give to a novice user, works wonders! It gets them going with a minimal amount of your time, and impresses the heck out of them – always a good start when dealing with novice users.

You should have readily available at all time a list of reference books. I have seen several posted to the net; an extensive one is posted regularly to `misc.books.technical` and other newsgroups.

My mother was a professional reference librarian for about 40 years, and when asked a "Do you know...?" question, her standard answer was "No, but I know where to look it up." The flip side to that, however, is that people *expected* her to look things up. Working in the music section of a large library, she was asked every day "When did Beethoven live?" If she instantly replied "1770 to 1827," the patron tended to be a bit suspicious. She found it necessary to pull out a book, flip through the pages, and refer to an authoritative source in order to convince the patron that that really was when Beethoven lived.

You should also have on hand easy-to-read information on getting started with netnews: advice on one or more news readers, and suggestions of groups to start with. If the user has a specific topic in mind, you might want to point them to one of the specialized groups, but you should think twice before inflicting one of the high-traffic groups on them. Then, of course, there are the man pages. I shouldn't have to say again how RTFM is an unhelpful suggestion, so let me just give a simple reminder: a good man page is designed for reference, not for teaching. Man pages are generally about one specific program, not the interactions of several programs. Think about the user's technical level of experience before handing them a multi-pound or multi-megabyte manual.

A Terrific Look-and-Feel

Somehow, the only times I feel as though I have the appropriate Look-and-Feel is when I'm either at a Usenix gathering or at a Grateful Dead concert. I used to work for AT&T, formerly known as The Phone Company. One aspect of that environment that never ceased to amaze me was that, despite being known world-wide for being at the forefront of communication and network technology, most of the users in my area preferred not to use the phone to ask me a question – they'd rather come to my office. This was not a small thing, since (by no doing on my part) my office was in the back corner at the far end of a long hall. I can't tell you how many times I've had a user walk down one hall, up a set of stairs, and down another hall to my room to ask me "What was that command to get the date?" Maybe they just needed the exercise.

Or maybe they preferred the look-and-feel of direct interaction with someone. Maybe the added element of voice communication, hand gestures and body language, access to but not reliance on a computer and other resources, maybe all these elements of the user interface made it worth their while to walk that distance.

A good look-and-feel doesn't mean that you have to wear a necktie, nor that you have to have scrollbars on your office door. It means having an attitude which says to users, "I am a knowledgeable person who can answer your questions." Such an attitude will inspire users to treat you as an expert, to treat you as a helpful source of information, and to treat you with respect – a treatment which is sure to improve your position in the community.

Conclusion

The key element of a well-designed user interface is that it lets the user get his job done without getting in his way. He isn't forced to jump through hoops to do a simple task, and he isn't prevented from doing a complicated task – in fact, he relies on the system to provide help of some sort at all levels.

As a system administrator, you are part of the resources available to the user. The way you interface with users affects the way they get their jobs done. With a little extra effort on your part, you can get them working at their best, and you'll all be able to take pride in the results.

Author Information

Wilson Bent recently joined the Research, Development, and Systems group of University Computing Services at the University of Southern California as a Systems Programmer. Previously, he worked for 10 years at AT&T Bell Laboratories as a programmer and administrator. Reach him via U.S. Mail at University Computing Services; University of Southern California; Los Angeles, CA 90089-0251. Reach him electronically at whb@usc.edu.

The System Administration Maturity Model – SAMM

Carol Kubicki – Motorola Cellular Infrastructure Group

ABSTRACT

The Capability Maturity Model (CMM), published by the Software Engineering Institute has been used by software development organizations to improve the processes used to develop software products.

The System Administration Maturity Model (SAMM) is an adaptation of the CMM to make it more relevant for system and network administration organizations. The SAMM seeks to describe the key processes required for a system and network administration organization to flourish into higher levels of process maturity and enjoy the benefits associated with mature organizations such as high quality products and services produced on time, and within budget limits.

Introduction

The motivation to apply process maturity concepts initially described in the Capability Maturity Model (CMM) to system and network administration comes from the many similarities between the two disciplines of software engineering and system and network administration. Although many disciplines struggle with issues regarding requirements, effort, manpower, budget, schedules, and quality, not all such disciplines work with highly inter-related subsystems where seemingly innocuous changes made by one individual impact the availability or performance of the entire system. The disciplines of network and systems administration and software engineering also share a common tendency to exploit the unique talents of single individual contributors.

Many of the common problems that are shared between system and network administration and software engineering are targeted for solution with the application of process maturity concepts. The application of process maturity concepts has already begun to pay dividends in software engineering organizations [6]. It is expect that the same types of results can be obtained from an emphasis on process maturity in the field of system and network administration.

A basic premise of process maturity efforts is that the quality of a product is determined by the quality of the process used to create it. High quality, robust processes applied in a network and systems organization will yield high quality products and services provided by that organization. Conversely, weak or immature processes will yield less than optimal products and services and results in an immature organization.

Characteristics of immature organizations include chaotic conditions, budget over-runs and schedule delays. Immature organizations often rely very heavily on the super-human effort of key individuals for success. In contrast, mature organizations are often characterized as producing highly reliable products and services in accordance with budget and schedule limits. The reliance on key individuals is replaced with an emphasis on a team atmosphere.

Process Maturity

The term *process* describes the means by which people, procedures, methods, equipment, and tools are integrated to produce a desired end result. The motivation to discuss process issues is derived from the fact that focus on singular components of the whole process will not yield the most effective improvements. In the past, administrators themselves have heavily focused on tools and methods with specific attention to automaton. Are we automating the right thing? Not much attention has been paid to the integration of all of the components of the system administration process. Thus some of the improvements we have made have fallen short of our own expectations, or the expectations of groups such as our management or end users.

In order to finally reap the rewards of improvement efforts, a systematic approach is required. A supportive foundation where one improvement is the basis for another is necessary. In the past, efforts that focused on tools or methods might not have been supported by policies and procedures for example. The SAMM describes the path from chaotic conditions to optimized conditions.

Each level of maturity and the improvements described as being associated with that level builds on the previous level and improvements. The model describes a framework of standardized states and conditions. Each organization must define how to achieve these states or conditions to eventually move to the next maturity level. Obviously, as system administrators support a wide variety of users, and the organizations that employ us have a wide range of priorities, the different processes created by each network and system administration group will vary widely. That these various implementations will vary should not detract from the fact that the implementations are based on the same framework of states and conditions.

It is important to note that, for the most part, problems facing network and system groups are not technical. Issues of communication, commitments, priorities and politics are common problems at most sites. Often problems such as these contribute to our personal frustration and organizational ineffectiveness to a much greater degree than technical problems do.

Historical Basis for Maturity Model

As described above, the SAMM is based on the CMM. It is useful to note however that the CMM is rooted in the quality control movement that began in the 1930's with the concepts of statistical quality control for primarily manufacturing operations. Inspired by the work of Deming and Juran, Philip Crosby offered a maturity framework for quality management. This framework describes five evolutionary stages in adopting quality practices. Adaptation of this framework to software development practices inspired the initial version of the CMM. These practices have once again been adapted to network and system administration to create the SAMM.

Quality management concepts have been refined as they have moved through the various disciplines beginning with manufacturing. Refinement is required as each new discipline such as software engineering presents greater problems for process improvement efforts in terms of complexity, technological issues, and coordination.

Use of SAMM

The SAMM can be used to create a road map for process maturity in system and network administration groups. The model can be used to create action plans for organizational improvement and excellence. The SAMM also provides a framework for assessment and evaluation.

Sites that might consider using the SAMM should be warned that this model is created for very large organizations. A small group with only a few members might find this model to be too cumbersome. This is not to say that SAMM is useless to smaller organizations, simply that difficulty to apply SAMM to a smaller group is acknowledged. It is also acknowledged that this model might appeal most to large corporate sites.

Organizations that are expected to enjoy the greatest success with SAMM are those where executive management is supportive of the concept of process improvement for all groups at the site (the network and system administration group and the user base as well as any intermediate organizations such as facilities and maintenance). In an environment such as this, it is expected that cooperation levels between the users, the intermediate organizations, and the network and system administration group will be much higher because all parts of the organization are working to achieve the same objectives.

Structure of SAMM

Key process areas are associated with one of five maturity levels. Organizations are all assumed to be level one organizations until they have been assessed to be higher. Assessment involves demonstration of organizational competence in key process areas. Eighteen key process areas are described with each having from two to fifteen key process activities. Competence in a process area involves commitment to the key process activities. The process areas are grouped to be associated with a single maturity level. The maturity level is a plateau on the path to higher maturity levels. Each maturity level must be achieved in numerical order to provide a solid foundation for later levels. Levels should not be skipped. See Table 1.

The fire fighting analogy in Table 1 suggests a simply way to relate to the various maturity levels. At level one the network and systems group is in crisis mode. A fire is being fought, but the battle is being lost. At level two, the network and systems group is fighting fires, but learning lessons during the fire fight about good, effective techniques. At level three, the organization enters a state of fire appreciation. It is acknowledged that there are fires,

Level	Title	Characterization	Analogy
1	Initial	Ad hoc process	Fire fighting – loosing
2	Repeatable	Disciplined process	Fire fighting – lessons
3	Defined	Standard, consistent process	Fire Appreciation
4	Managed	Predictable process	Fire Detection
5	Optimizing	Continuously Improving	Fire Prevention

Table 1: Five Maturity Levels

and the organization begins to understand the unique characteristics of different types of fire. At level four, focus shifts to detection of smaller fires before they evolve into major fires. At level five, fire prevention is the main theme.

Descriptions of terms

Champion – The term champion is used often to identify one or more key individuals who take responsibility for various facets of process improvement. These champions can be internal such as members of the network and systems administration group, or external such as users or members of other interested organizations like a quality assurance organization. Specific champions described are champions of quality assurance, technological change, and process improvement. In the event that experts in the areas of quality, technology, or process are available outside the network and systems group they should be utilized to provide an objective view.

Customer – The term customer is used when referring to the actual paying customer of a site. This would be the person or organization that purchases the products or services created by the supported organization.

Engineering – The term engineering is often used when describing the network and systems group. This an intentional usage of the term to bring attention to the fact that system and network administration is an engineering discipline.

Implementation – The term implementation is used to refer to the actual solution deployed by the network and systems administration group to meet requirements. The implementation could be a product, service or activity.

Life Cycle – The term life cycle refers to the period of time beginning with the conception of an idea for a network and systems related activity, product or service and ends when that activity, product or service is no longer available for use.

Network and systems group – This is the default term used to describe the system administration organization. It is understood that not all such organizations have responsibility for networking related functions. Ideally, SAMM users can put the name of their own organizations in place of the term network and systems group.

Supported organization – The term supported organization is used to describe the organizations that consume the products and services of the network and systems administration group. Ideally both parties, the supported organization and the network and systems engineering group are aware of their relationship with each other.

Key Process Areas & Activities

The key process areas are described below. Each description includes a title, abbreviation, association with the correct maturity level and sections for purpose, requirements, goals and discussion.

Level	Characterization	Key Process Area
2	Repeatable	Requirements Management
		Project Planning
		Project Tracking
		Subcontract/Vendor Management
		Quality Assurance
		Configuration Management
3	Defined	Process Focus
		Process Definition
		Training
		Integrated Management
		System & Network Engineering
		Intergroup Coordination
		Peer Review
4	Managed	Quantitative Process Management
		Quality Management
5	Optimizing	Defect Prevention
		Technology Change Management
		Process Change Management

Table 2: Process Areas by Maturity Level

Key process activities are numbered for easy reference.

For example, the first key process area is Requirements Management (RM) which is associated with the repeatable level (level 2). There are two activities associated with the RM area RM-1, and RM-2. See Table 2.

Requirements Management (RM) – Repeatable

Purpose: to generate and document a common understanding between the supported organization and the network and systems group of the requirements that will be addressed by the network and systems effort.

Requires: the creation and maintenance of an agreement with the supported organization.

Goals: Plans and activities are kept consistent with the agreed upon requirements.

Key Process Activities

1. The network and systems group reviews documented requirements before they are accepted as commitments or presented to the supported organization as commitments.
2. The network and systems group uses the reviewed requirements as the basis for plans, activities and products and/or services.

Discussion: Requirements are the foundation for most types of activities providing a clear description of what is expected by the supported organization. Most commonly, requirements documents are associated with large scale projects. However, requirements can be written for services and activities to clarify expectations also. In addition to functionality or feature requirements, other issues such as performance, user interface, and documentation should be considered as items to cover during requirements gathering.

Project Planning (PP) – Repeatable

Purpose: to establish plans for performing network and system effort and managing the activities associated with projects, products, and/or services.

Requires: estimates, commitments, definition, reviewed requirements.

Goals: Document an agreement of estimates, activities, and commitments.

Key Process Activities

1. The network and systems group participates in the project planning activities of the organizations they support (from beginning to end).
2. Network and systems planning is initiated in the early stages of, or in parallel with, the planning activities of the supported organizations.
3. Commitments regarding plans, activities, products, and services are reviewed with network and systems management prior to being made to the supported organizations.
4. A life cycle with predefined stages is identified for use.
5. The plan is developed according to a documented procedure.
6. The plan is documented.
7. Information or equipment required to maintain control of the project or activity is identified.
8. Estimates regarding the scope of the project or activity are derived according to a documented procedure.
9. Estimates for the effort and cost of the project or activity are derived according to a documented procedure.
10. Estimates for computing/networking resources (hardware and facilities) are derived according to a documented procedure.
11. The project schedule is derived according to a documented procedure.
12. Project risks are identified, assessed, and documented.
13. Plans to acquire required software resources (internally developed or purchased software – operating system and application software including customization) are derived.
14. Project planning data is recorded.

Discussion: Project planning within the network and systems group can be driven by the supported organizations when they require products or services from the network and systems group, or undertaken internally when the network and systems group determines that a particular action is appropriate. A main point here is that many projects undertaken by the supported organizations at a site require products or services from the network and systems group. Therefore, the network and systems group should participate in the early planning of such activities undertaken by the supported organizations. Early involvement will allow for better communication and allow time for both organizations to develop a project plan. Example projects and activities which might originate in the supported organization are departmental moves and new production, build or release schedules.

Another benefit achieved through participating in the development of project plans for the supported organization is visibility to the clear statement of goals for projects and activities found in the plan, as well as, statements of responsibility for the activities described in the plan.

Project Tracking (PT) – Repeatable

Purpose: to monitor actual performance of the network and systems group relative to estimates documented in the plan.

Requires: tracking and review of accomplishments and results

Goals: Track actual results of the network and systems group against the plan, and take corrective action when required. Effected groups agree to commitment changes if required.

Key Process Activities

1. A documented plan is used for tracking the project activities and communicating status information.
2. Revisions to the project plan are made according to a documented procedure.
3. Commitments and changes to commitments are reviewed with management according to a documented procedure.
4. Changes in the supported organization that might impact the network and systems organization are communicated to the network and systems group.
5. Actual scope of the project or activity is tracked (some measure of size and/or complexity). Corrective action is taken when required.
6. Actual effort and costs of the project or activity are tracked and corrective actions are taken when required.
7. The schedule documented in the plan is monitored, and corrective action is taken when required.
8. Technical activities are tracked including status information and problem resolution details.
9. Risks associated with cost, resources, schedule and technical aspects of the project or activity are tracked.
10. Replanning data is recorded.
11. Reviews are conducted within the network and systems organization to track technical progress, plans, performance and issues against the project plan.
12. Formal reviews to address the accomplishments and results of the project are conducted at selected milestones according to a documented procedure.

Discussion: Successful project tracking allows the network and systems organization to identify schedule and resource issues before major deadlines or commitments are missed. Part of project tracking involves information moving both to and from the supported organizations. Project review meetings can be held with or without the attendance of representatives of the supported organization as appropriate.

Subcontract and Vendor Management (SVM) – Repeatable

Purpose: to select qualified subcontractors and vendors and manage them effectively.

Requires: selection of subcontractors and vendors, establishing commitments and relationships with the subcontractors and vendors including review of performance and results.

Goals: Communication regarding mutual commitments between the network and systems group and the subcontractors and vendors. Track actual results against commitments and feed information back to subcontractors and vendors.

Key Process Activities

1. Activities to be subcontracted or addressed by vendors are defined, planned, and documented according to a documented procedure.
2. Subcontractor and vendors are selected based on an evaluation of their ability to provide products and/or services according to a documented procedure.
3. The agreement between the network and systems organization and the subcontractor/vendor is used as the basis for managing the relationship.
4. Changes to the agreement or activities are resolved according to a documented procedure.
5. Management reviews are conducted periodically between the managers of the network and systems organization and the management of the subcontractor or vendor.
6. Technical reviews are conducted with the subcontractor or vendor to encourage communication (Reviews of new products and contractor skills or abilities).
7. Formal reviews to address the accomplishments of the subcontractor or vendor and results including service performance and/or new products are conducted at specified phases (of a project or calendar cycle) according to a documented procedure.
8. The quality assurance activities of the subcontractor or vendor are monitored according to a documented procedure.
9. The configuration management activities of the subcontractor or vendor are monitored according to a documented procedure.
10. The network and systems group conducts acceptance testing as part of the delivery of the products and/or services provided by the subcontractor or vendor according to a documented procedure.
11. The performance of the subcontractor or vendor is evaluated on a periodic basis and the evaluation is reviewed with the subcontractor or vendor.

Discussion: Vendors and subcontractors play a key role in most network and system groups. Vendors supply most of the hardware and software used to build and maintain systems and networks. They are expected to honor commitments related to prices, delivery schedules, bug fixes, and compatibility issues. Hardware and software vendors are also expected to provide new technologies to address the needs (described or unknown) of the supported organizations. Subcontractors are the various consultants or experts we might hire on an as needed basis or for outsourcing support of specific functions. Other groups that should be considered subcontractors are the organizations at the site chartered to provide services to the network and systems group such as a facilities or physical plant organization or even a finance or purchasing group.

Quality Assurance (QA) – Repeatable

Purpose: to provide visibility to the process being used and the results being achieved by the network and systems organization.

Requires: reviews and audits and communication of results of reviews and audits.

Goals: Compliance of activities, products and services to applicable standards, procedures and requirements is verified objectively.

Key Process Activities

1. A quality assurance plan is prepared for the project or activity according to documented procedures.
2. Quality assurance activities are performed in accordance with the Quality assurance plan.
3. Quality assurance champions participate in the preparation and review of the project plan, standards, and procedures.
4. Quality assurance champions review the activities of the network and systems group to verify compliance to project plans and standards.
5. Quality assurance champions report results of audits and reviews to the entire network and systems group.
6. Deviations from plans and standards are documented and addressed according to a documented procedure.

Discussion: The theme of this key process area is the collection of information. Quality assurance activities will heighten the level of quality consciousness in the network and systems group through routine feedback of quality related information. Participation of champions who are quality focused as opposed to technically focused in the project planning efforts will bring greater attention to quality issues to be considered early in the design of products and services.

Configuration Management (CM) – Repeatable

Purpose: establish and maintain integrity of the activities, products, and services through the entire life cycle.

Requires: identification of configuration items and systematic control of changes to any configurable items.

Goals: Changes to identified activities, products, and services are controlled.

Key Process Activities

1. A configuration management plan is prepared for each project or activity according to a documented procedure.
2. A documented configuration management plan is used for the basis of configuration management activities.
3. A library is established as a repository for configurational information including status information and change justifications.
4. The items to be placed under configuration management are identified.
5. Change requests against configuration items are initiated, recorded and tracked according to a documented procedure.
6. Products and services created and released by the network and systems group are controlled according to a documented procedure.
7. Reports documenting configuration management activities are developed and made available to effected groups and individuals.

Discussion: Network and systems groups have many configurational items that are candidates for configuration management. Various system files, application software, OS revisions, kernels, and network diagrams are just a few. Robust configuration management of these and other items will help ensure that changes that are made are justified and the release of these changes is coordinated. The integrity of the changes can also be improved with configuration management practices.

Network and Systems Process Focus (PF) – Defined

Purpose: Champion the process activities that improve the overall process capability of the network and systems group.

Requires: an understanding of the network and systems group processes and initiating activities to assess, develop, maintain and improve these processes.

Goals: Process development and improvement activities are coordinated across the network and system group including assessing the strengths and weakness of various processes.

Key Process Activity

1. Network and systems processes are assessed periodically, and action plans are developed to address the assessment findings.
2. The network and systems group maintains a plan for process development and improvement activities.
3. New processes, methods, and tools used in parts of the network and systems group are monitored, evaluated and if appropriate released to the entire group.

4. Training in network and systems processes is coordinated.
5. The entire network and systems group is kept informed of process improvement related activities.

Discussion: As the network and systems group moves to higher levels of maturity, processes will naturally be refined. The process focus of the group is required to continue improvements. It is also necessary to have process champions available to consult within the network and systems group on process issues and conduct assessments as needed.

Network and Systems Group Process Definition (PD) – Defined

Purpose: To develop and maintain processes and improve process performance across the projects, services and activities of the network and systems group.

Requires: collection and release of organizationally significant process related information to the network and systems group.

Goals: Standard processes for the network and systems group are created and maintained. Information related to the use of standard processes is collected and made available.

Key Process Activity

1. Network and systems group standard processes are developed and maintained according to documented procedures.
2. Descriptions of project life cycles that are approved for use by the network and systems organization are documented and maintained (phase overlap, waterfall).
3. Guidelines to tailor the standard processes are developed and maintained.
4. A process database of information such as quality and productivity data is created and maintained.
5. A library of process related documentation is created and maintained.

Discussion: As the maturity level of the network and systems group increases, it is necessary to support the refinement of processes with the general availability of all process related information. Both general information such as reference materials and project specific information such as quality and productivity data should be made available.

Training – (T) Defined

Purpose: to develop skills and knowledge of individuals to enable them to perform roles effectively and efficiently.

Requires: identification of the training needs of the networking and systems group and development or acquisition of training to meet identified needs.

Goals: Appropriate training is provided to the right people at the right time.

Key Process Activity

1. Training plans outlining the training needed by the network and systems group as a whole, or the various project or activity teams are created and maintained.
2. The training plan is developed and revised according to documented procedure.
3. The training of the network and systems group is acquired in accordance with the training plan.
4. Internal training courses are developed and maintained according to a documented procedure.
5. An assessment process is developed to determine if necessary skills are already possessed by the staff or should be acquired through training.
6. The network and systems group maintains a record of all staff training.

Integrated Management (IM) – Defined

Purpose: to integrate the engineering of network and systems related products, activities, and services with management activities to move from simply tracking problems to anticipating problems.

Requires: a project plan and standard organizational processes.

Goals: The project and activities are planned and manged according to the defined processes.

Key Process Activity

1. Standard processes are tailored according to a documented procedure.
2. Project plans are developed and revised according to a documented procedure.
3. Projects and activities are managed in accordance with the defined process.
4. The process database is used as a source of information for planning and estimating.
5. The scope of the activities or project is managed according to a documented procedure.
6. The effort and cost associated with a project or activity are managed according to a documented procedure.
7. Use, availability, and/or performance of computing hardware, software, and/or networking resources are managed according to a documented procedure.
8. Critical dependencies and critical paths identified in the project plan are managed according to a documented procedure.
9. Project risks are identified, assessed, documented, and managed according to a documented procedure (including risk indicators and early identification of risks).

Discussion: Activities associated with integrated management are feed by the previously described project tracking processes and the standard organizational processes. The goal is to use historical project tracking information to begin to anticipate problems and either prevent them or minimize their effects.

System and Network Engineering (SNE) – Defined

Purpose: to perform a well defined engineering process that integrates all network and systems engineering activities to produce correct and consistent products and services efficiently and effectively.

Requires: requirements, design, implementation, integration, testing

Goals: Network and systems related tasks and activities are defined, and consistently performed as required to produce consistent products and services.

Key Process Activity

1. Appropriate engineering methods and tools are integrated into the defined network and systems process.
2. The requirements of a supported organization are developed, maintained, documented and verifyed by analyzing them according to the defined network and systems process.
3. The design for a project or activity is developed, maintained, documented, and verified according to the defined network and systems processes to accommodate requirements and to form the basis for implementation.
4. The implementation is developed, maintained, documented, and verified according to network and systems engineering processes to meet requirements and design goals.
5. Testing is performed according to network and systems processes.
6. Integration testing of the implementation is planned and performed according to network and systems processes.
7. System and acceptance testing of the implementation is planned and performed to demonstrate that the implementation satisfies its requirements.
8. Documentation used to operate and maintain the implementation is developed and maintained according to the network and systems processes.
9. Data on defects identified in peer reviews and testing are collected and analyzed according to network and systems processes.

Discussion: The system and network engineering process area brings greater attention to the sound engineering practices (such as analysis, test and acceptance) that are required to create and maintain the products and services associated with the network and systems group.

Intergroup Coordination (IC) – Defined

Purpose: to establish a means for the network and systems group to participate actively with other groups in the organization to ensure that the activities, products, and services are best able to satisfy the customer needs effectively and efficiently.

Requires: disciplined interaction and coordination

Goals: To reach agreement of requirements, commitments and priorities with other groups.

Key Process Activity

1. The network and systems group participates along with the supported organization and the customer or end users of the supported organization in establishing requirements where appropriate.
2. Representatives of the network and systems engineering group work with the supported organization and other intermediate organizations to coordinate technical activities and resolve issues.
3. A documented plan is used to communicate intergroup commitments and to coordinate and track the commitments and activities.
4. Critical dependencies between groups are identified and tracked according to a documented procedure.
5. Intergroup issues not resolvable by the individual representatives are handled according to a documented procedure.
6. The supported organizations conduct periodic technical reviews and interchanges to provide visibility of the needs of the end customer.

Discussion: The intergroup coordination process area focuses on the ability of the network and systems group to address customer needs. In many cases, the supported organization produces products or services that could be enhanced with the application of network and systems group effort. Only if the network and systems group is aware of these opportunities can they meet the needs of the end customer.

Peer Reviews (PR) – Defined

Purpose: to remove defects from network and systems group activities, products and services early and efficiently.

Requires: examination of the activities, products and services by peers to identify defects.

Goals: Defects in activities, products and services are removed.

Key Process Activity

1. Peer reviews are planned, and the plans are documented.
2. Peer reviews are performed according to a documented procedure.

3. Data from the peer review is recorded.

Discussion: Products, activities, and services subject to peer review are identified in the network and systems engineering processes. Time for reviews is scheduled in the project plan. Reviews do not need to be limited to the traditional code review where specific lines of code are examined. Network and systems group activities, products, and services can be reviewed in a peer setting using a variety of other means such as role play, simulation, document reviews and test cases.

Quantitative Process Management (QPM) – Managed

Purpose: quantitatively control the process performance of network and systems group processes.

Requires: establishing goals, measuring against goals, analysis of measurements.

Goals: The process capability of the network and systems group is known in quantitative terms.

Key Process Activity

1. The plan for quantitative process management is developed according to a documented procedure.
2. The network and systems group process management activities are performed in accordance with the quantitative process management plan.
3. The strategy for data collection and analysis is determined based on network and systems group processes.
4. The data used to control defined software process quantitatively is collected according to documented procedure.
5. Network and systems group processes are analyzed and brought under quantitative control according to a documented procedure.
6. Reports documenting the result of quantitative process management activities of the network and systems group are prepared and distributed.
7. The process capability baseline for network and systems group processes is established and maintained according to documented procedures.

Discussion: The emphasis in this process area is on the quantitative results of the processes used by the network and system group. Data is collected to characterize the capability of the network and systems group processes. Process capability describes the range of results that can be achieved by following a specific process. Capability information is used within the network and systems group to adjust process performance goals for future activities, products, and services.

Quality Management (QM) – Managed

Purpose: to develop a quantitative understanding of the quality goals for the network and systems group products, activities, and services.

Requires: defined quality goals and plans to achieve quality goals.

Goals: Establish measurable goals for quality levels and priorities. Progress to achieve quality goals is quantified and managed.

Key Process Activity

1. Network and systems group quality plans are developed and maintained according to a documented procedure.
2. The network and systems group quality plans are the basis for quality management activities.
3. Quantitative goals for products, activities and services are defined, monitored and revised throughout the life cycle of the product, activity or service.
4. The quality of network and systems group products, activities and services is measured, analyzed, and compared to quantitative quality goals throughout the life cycle of the product, activity, or service.
5. The quantitative quality goals of the network and systems group are shared with, or responsibility assigned to, network and systems group subcontractors or vendors as appropriate.

Discussion: Quality management practices build on the quality assurance practices of the network and systems group. The focus shifts form data collection to the management of the quality of the activities, products and services in quantitative terms.

Defect Prevention (DP) – Optimizing

Purpose: to identify the cause of defects and prevent them from recurring.

Requires: historical defect information from similar activities, products, and services, as well as, defect information from early stages of design or testing of a given activity, product, or service.

Goals: Common causes of defects are sought out, prioritized, and eliminated.

Key Process Activity

1. The network and systems group develops and maintains a plan for defect prevention activities.
2. At the beginning of a task or activity, the members of the networking and systems group meet to prepare for the task or activity and the related defect prevention activities.
3. Causal analysis meetings are conducted according to a documented procedure.
4. Causal information from various tasks and activities is reviewed at the by the entire

network and systems group periodically to share information and set priorities.
5. Defect prevention data is documented and tracked across the network and systems group.
6. Revisions to the network and systems group processes resulting from defect prevention activities are incorporated according to documented procedures.
7. Defect prevention information (status and results) is shared with the entire network and systems group periodically.

Discussion: Root cause analysis is key to defect prevention. Obviously if several different defects are caused by an identifyable and preventable root cause, these detected defects can be eliminated in the future from any products or services that might utilize the effected item.

Technology Change Management (TCM) – Optimizing

Purpose: to identify new technologies and move them into the network and systems group in an orderly manner.

Requires: identification, selection, and evaluation of new technologies.

Goals: New technologies are evaluated for their effect on the quality and productivity of the network and systems group, as well as, the supported organizations. Appropriate new technologies are integrated.

Key Process Activity

1. The network and systems group develops and maintains a plan for technology change management.
2. Champions of technological change work with the entire network and systems group to identify areas of technology change.
3. The entire network and systems group is kept informed of new technologies.
4. Champions of technological change analyze the network and systems group processes to identify areas that need or could benefit from new technology.
5. Technologies are selected and acquired for the network and systems group according to a documented procedure.
6. New technologies are tested before a new technology is introduced into routine use in the network and systems group.
7. New technologies are incorporated into the network and systems group processes according to documented procedures.

Discussion: Technological change is a major issue for most network and system groups. This key process area focuses on those technologies that will advance the levels of quality and/or productivity in the network and systems group. These technologies might directly or indirectly benefit the supported organization when they are integrated.

Process Change Management (PCM) – Optimizing

Purpose: to continually improve the network and systems processes with the intention to improve quality, increase productivity, and decrease cycle time for activities, products and services.

Requires: management sponsorship to proactively identify, define, and implement improvements to network and systems group processes.

Goals: continuous, organization wide process improvement.

Key Process Activity

1. A process improvement program is established to empower all members of the network and systems group to improve network and systems processes.
2. Champions of process improvement coordinate network and systems group process improvement activities.
3. The network and systems group develops and maintains a plan for process improvement according to a documented procedure.
4. Network and systems group process improvement activities are performed in accordance with the documented process improvement plan.
5. Network and systems process improvement proposals are handled according to a documented procedure.
6. Various members of the network and systems group actively participate in the development of process improvement ideas for specific process areas.
7. If possible, network and systems process improvements are tested on a pilot basis to determine their benefits and effectiveness before they are introduced into routine use.
8. When an improvement is moved from pilot use to routine use, the improvement is deployed according to documented procedures.
9. Records of network and systems group process improvement activities are maintained.
10. The entire network and systems group is provided with process improvement information including status and results on an event driven basis.

Discussion: Continuous improvement at every level of the organization is sought. Even the processes that allow the network and systems group to mature can be improved. Process change management seeks to bring process innovations into the network and systems group in a controlled way to encourage continuous process improvement.

Common Practices by Maturity Level

A few common system and network administration practices are described below with reference to the various maturity levels. The intention is not to say that accomplishment of any or all of these tasks as described will place an organization at the specific level. This mapping is simply to show some of the changes required of the higher level key process areas.

Level 1 – Initial

New user – Verbal requests are addressed as time permits or escalated with management.

Software install – New or upgraded software is installed whenever & where ever makes the most sense at the moment.

Hardware install – New or upgraded hardware is installed whenever & where ever makes the most sense at the moment.

Problem report – Problems are sometimes reported by users by mail or phone to a random administrator.

Security – No specific security standards or policies exist.

Disk capacity – Disk space is in short supply. No information on usage rates is available. Project managers of supported organizations often fight among themselves regarding disk space.

Backups – Backups are usually done according to a weekly schedule.

Level 2 – Repeatable

New user – Procedure to request and create an account is well documented. Cycle time for requests is monitored.

Software install – Installation guidelines are understood. Time spent installing and configuring software is tracked.

Hardware install – Installation standards are understood. Time spent installing and configuring hardware is tracked.

Problem report – Process to report problems is well understood by users and cycle time for problem resolution is monitored.

Security – Various security standards are clearly documented. Security violations are monitored.

Disk capacity – Acceptable disk capacity levels are established. Capacity is periodically monitored.

Backups – Failure conditions for backups are understood. Failure rates and effort to resolve problems are tracked.

Level 3 – Defined

New user – Head count expansion information is provided to signal new account requests are expected. Revision of procedure.

Software install – Installations are planned with the supported organizations Reasons for install/upgrade are recorded.

Hardware install – Installations are coordinated with supported organizations and vendors. Reasons for install/upgrade are recorded.

Problem report – Problems are mapped to root causes. Group reviews solutions to resolutions suggested to address root causes.

Security – Group reviews security incidents for root vulnerabilities. Resolutions are discussed, tested and released.

Disk capacity – Capacity planning is addressed in project plans written by supported organizations and reviewed by the network & systems group.

Backups – Training is provided for network and systems group to enhance backup programs and procedures.

Level 4 – Managed

New user – Cycle time numbers are used to adjust staffing to meet demand and requirements.

Software install – Productivity measures and goals for installation created.

Hardware install – Productivity/problem data is compare to goals for installs, tests and demos.

Problem report – Cycle time and resolution quality information is used on a regular basis to access effectiveness of problem reporting and resolution system.

Security – Effectiveness of group reviews is studied.

Disk capacity – Metric for disk availability is created and used.

Backups – Backup system is certified with high reliability rating.

Level 5 – Optimizing

New user – Accounts are electronically requested and verified.

Software install – Problems with software installs are documented and avoidable with new procedures.

Hardware install – New hardware technologies are evaluated & integrated.

Problem report – New problem reporting system installed to better meet user requirements for ease of use.

Security – Internal contest to establish better security practices is established.

Disk capacity – Project tracking information combined with metrics of utilization are used to predict needs.

Backups – Backup process is revisited with input from supported organizations re: production schedules.

Instrumentation

The following method can be used to roughly estimate the maturity level of a network and systems group within the scope of the SAMM.

Each key process activity is examined and a single score awarded for the entire organization being assessed. In the case of the Requirements Management key process area (RM) there are two key process activities. A score is determined for each activity in the key process area and the average of those two scores is the score of the area. The matrix below lists the characteristics associated with various scores (0, 2, 4, 6, 8, and 10). Borderline scores (situations that fall between those specified) can also be awarded (1, 3, 5, 7, and 9).

To describe a network and systems group as being at the repeatable maturity level (level two), all key process areas associated with that level must be qualified or receive an area score of 8. See Table 3.

Conclusions

The SAMM process improvement framework presents a challenge to the field of network and system administration. In the past, some might have considered it a matter of pride to think of system administration as some sort of magic and consider our processes to be a collection of black boxes. In order to advance the state of our profession, it is necessary to diagram and document those familiar black boxes.

Acknowledgement

The Capability Maturity Model (CMM) is developed and maintained by the Software Engineering Institute (SEI) of Carnegie Mellon University as a tool to assist software development organizations with developing higher quality software. Simply listing this work as a reference doesn't seem appropriate given the fact that it is the entire basis for this paper.

Author Information

Carol Kubicki is employed by Motorola's Cellular Infrastructure Group in Arlington Heights, Illinois as a Senior Network and Systems Engineer. Carol is currently serving on the SAGE board of directors and is working toward a Masters in Management and Organizational Behavior where her interests include quality and organizational culture. Reach her via U.S. Mail at Motorola; 1501 West Shure Drive; Arlington Heights, IL 60004. Reach her electronically at kubicki@mot.com.

Rating	Score	Characterization
Poor	0	No ability No interest Ineffective results
Weak	2	Partial ability Fragmented usage Inconsistent results
Fair	4	Implementation Plan defined Usage in major areas Consistent positive results
Marginal	6	Implementation across organization Usage in most areas Positive measurable results
Qualified	8	Practice is integral part of process Consistent use across organization Positive long term results
Outstanding	10	Excellence in practice well recognized Consistent long term use Consistent world class results

Table 3: Evaluation Matrix

References

[1] Crosby, P. *Quality is Free*. McGraw-Hill, New York, NY. 1979.

[2] Motorola Corporate Engineering Council. *Product Development Assessment Guidelines*. Internal document of Motorola, Schaumburg, IL 1992.

[3] Paulk, M., Curtis, B., Chrissis, M., and Weber, C. *Capability Maturity Model for Software, Version 1.1*. Software Engineering Institute, CMU/SEI-93-TR-24, February, 1993.

[4] Paulk, M., Bush, M., Chrissis, M., Garcia, S., and Weber, C. *Key Practices of the Capability Maturity Model, Version 1.1*. Software Engineering Institute, CMU/SEI-93-TR-25, February, 1993.

[5] Paulk, M.C. *U.S. Quality Advances: The SEI's Capability Maturity Model*. Internal document of the Software Engineering Institute, Pittsburgh, PA 1992.

[6] Quantitative Software Management. "Project Data" (data collected from 2,800 software development projects). McLean, VA 1993.

[7] Von Mayrhauser, A. *Software Engineering Methods and Management*. Academic Press, San Diego, CA 1990.

Establishing and Administering a Public Access Internet facility

Sheri Byrne – Gemini Learning Center

ABSTRACT

In a perfect world, Internet access should be available to anyone who wants it. In reality, that is not an easily achievable goal. In many geographical locations there are "haves" – people with Internet access via large corporations or academic connections and "have nots" – contractors, consultants, and employees of small companies with no way of accessing the Internet.

The Calgary Unix Users Group (CUUG) Computer Resource Centre (CRC) was established in March 1991 to provide UNIX facilities and Internet connections to its members. This paper describes the evolutionary process that went into the conception, establishment, and current operation of the CUUG CRC (Internet address cuug.ab.ca). This includes both technical issues such as physical administration of the systems, and non-technical (but equally important) issues such as coordinating volunteer and fundraising activities and legal ramifications.

Motivation

Calgary, Alberta has the third largest base of UNIX workstations in North America (DMR, 1988), yet most Albertans had little or no access to the Internet prior to 1991. All access to the Internet in the province of Alberta is controlled through the Alberta Research Council (ARC), a government-subsidized, high-technology corporation, who charge a substantial yearly fee for corporate Internet access, significantly less for non-profit access. The only other method of Calgary Internet access in 1989 without incurring long distance charges was through the University of Calgary, which at that time had limited non-student access, which has since been suspended.

Some Calgary-based uucp feeds were available which supported mail and netnews access. However, these were considered to be too "home-grown" for any type of serious usage. Phone line access was limited, down time was considerable, and the majority of these establishments were run out of basements as hobbies and bulletin boards rather than as full-time, business-oriented enterprises.

The need for dissemination of UNIX information was evidenced by the establishment of the Calgary UNIX Users' Group in September, 1990. Less than three months after it began, CUUG boasted a membership of over 300. CUUG's primary goal is to encourage the use of UNIX and open systems. In January, 1992 the CUUG Membership Services Committee decided to partially fulfill this goal by providing Internet access to its membership.

Scope

The CUUG CRC currently consists of five ethernetted UNIX platforms, an Intel-based 486 which is owned by CUUG, the others (IBM RS6000 320, Silicon Graphics Indigo, HP 9000/730 and Sun Sparc 2) on loan from hardware vendors (see Figure 1 for network layout details). The system currently has fourteen telephone lines. Two of these lines are dedicated to the University of Calgary, one line is unlisted for administrator use only, and eleven lines are for public access. There is a Gandalf terminal server on the front end to transfer login access to the specified UNIX system via TCP-IP. The CRC performs the following functions:

1) Access to Internet e-mail, ftp, telnet, net news
2) Archive site for exchange of newsletters amongst UNIX users group
3) Application porting facility for both non-UNIX and UNIX software programs to other UNIX platforms.

The CUUG CRC had to be solid enough for long-term, business-oriented projects like software ports. However, the setup costs of $8900 CDN and initial yearly budget of $11,000 CDN did not provide resources for a paid system administration staff of any kind.

Hardware Logistics

Originally, many people thought that acquiring the hardware necessary to run such a system would be a major stumbling point. However, this turned out to be the least of the logistical problems. This was accomplished by tying the CRC hardware with the CUUG Corporate Sponsorship program. In the first year, only Gold level sponsors were allowed to loan hardware to the CRC, in succeeding years, hardware loan is now a requirement for Gold Sponsorship status.

It was determined that a long-term loan, rather than donation, was the preferable method of obtaining the hardware. This bypassed potential problems and costs associated with insurance and service contracts, since the sponsors still owned the systems.

Since its inception, CUUG has had a waiting list for gold corporate sponsor status. However, to guard against the day CUUG did not have such a luxury, a 486 running Interactive Unix was purchased, to guarantee that there would always be at least one server available to provide Internet access.

One of the side-effects of the "hardware loan" arrangement is system volatility. In addition to the six hardware vendors currently represented, hardware from DEC, Bull and AT & T has also been part of the CRC over the past 2 1/2 years. There have been several occasions where systems have been pulled or swapped on very short notice due to the vendors desire to upgrade to a better system, or need to sell an asset to avoid tax implications. Therefore, CRC administrators have been forced to make many decisions with this hardware volatility in mind.

One decision influenced by this is the need to make only full filesystem dumps, for quick disk restoration. It was not feasible to obtain an 8mm or DAT tape drive from every sponsor, therefore, CUUG's second major hardware investment was an Exabyte 8mm tape drive which was put on the CUUG-owned 486 and networked to the other systems.

Location Logistics

The CRC had to be located in a vendor-neutral setting which was easily accessible by the members who wished to do on-site work which required X Window Systems support, such as graphical user interface development. This ruled out vendors themselves, VAR's of vendors, and anyone who wasn't in the downtown core, where the majority of Calgary computer-oriented business is conducted. In addition, the space had to be free, and people who knew rudimentary UNIX system operation had to be available in case of a system emergency.

The CUUG CRC hardware was eventually installed at the offices of Gemini Learning Systems, Inc. Five isolated ground circuits and five telephone lines (eventually expanded to fourteen) were added to support CRC system operation requirements. The CUUG CRC systems are available 24 hours/day, 365 days a year via modems ranging from 2400 baud to 14.4 Kbaud, and during normal weekday business hours.

Administration Personnel

The logistics of adminstering the hundreds of users on the five CUUG CRC systems was compounded by the fact that it was done on an entirely volunteer basis. System setup and administration tasks were broken down into the following categories:

- News
- Mail
- UUCP feeds
- Software Installation
- Account Maintenance
- Administration which required physical system access (backups, hardware troubleshooting, manual system reboot, etc.)

and assigned to five different volunteers. Those who absolutely required root access were provided it; all others were added to the "helpers" group where they had privileges to install software. Any system work which required root access or rebooting was coordinated through the chairperson of the CUUG Membership Services Committee to avoid potential system conflicts.

Users were advised to send questions and requests to the postmaster, who forwarded them to the appropriate volunteer. A voice telephone line answered by a non-technical person was available for general system information and to report emergencies such as system crashes or modem outages.

The efforts of the five original volunteers were augmented by a "System Administrators Assistant" work / co-operative education program established between CUUG and the Southern Alberta Institute of Technology (SAIT) Computer Technology Department. Every semester, two SAIT students would work at the CUUG CRC doing day-to-day system administration tasks such as account maintenance and backups. Each student was responsible for developing an application sometime during the semester which automated part of the system administration process which was left as a legacy to future students and other CUUG volunteers.

Security

System administration tools (such as "cops" and "crack") were obtained and run regularly as a first line of security defense. In addition, several shell scripts were developed to aid in monitorng CUUG's unique requirements. These scripts were developed in Bourne Shell for portability.

It was illegal to possess any password cracking utilities on the CRC without first notifying one of the administrators that the files would be there temporarily, and notifying the system administrator of the destination machine they were eventually being transfered to. Anonymous ftp was restricted to one system only.

All attempts to "su" were tracked (see program "intruders"). Actual system usage at a process level was also tracked (see program "monitor") in order to generate reports to the hardware sponsors to let them know of both the quantity and nature of the traffic on their particular system.

Disk Management

CUUG wanted to provide the most flexible environment possible for their users. There is approximately 1.6 GB of disk space across the five systems for the users to share. After much discussion by the CUUG CRC volunteer administrators, it was decided to manage this disk space without implementing fixed quotas. In order to do so, it was necessary to be able to monitor where the disk space was being used, by user, and most importantly on an incremental basis (i.e. which users had recently used a large chunk of space)

A disk space monitoring program which reviewed current usage against the most recent report separated reported on who was using more than 1 MB of disk space (see program "disk_pig"). An adjunct program sent mail automatically (see program "hog_mail") to both root and the offending user. Users who requested disk space leniency for a particular project for a fixed period of time were excluded from the disk space policing, but their usage was monitored through quota.

When disk space got tight, there were checks made for multiple copies of common utilities like kermit or xdbx. The duplicated copies were compressed, the users were pointed to the system copy, and they were asked to remove their private copy. Core files were removed nightly, and accounts where the membership had lapsed were deactivated (see programs "starout", "deactivate", "delete_users") removed after several warnings and two months. Core files were deleted nightly (see program "corekiller").

After two years, due to the overwhelming amount of e-mail, a 100K limit was placed on both outgoing and incoming e-mail.

Shared logins, Use Limitation, and UUCP feeds

Because CUUG is a non-profit organization, the volunteer administrators discouraged users from sharing accounts with others in their companies. With only four public access lines in the initial configuration, it would not have been fair for one login to be re-used on all four lines, effectively blocking other equal paying members from using the system.

This was originally established as a recommendation, and eventually after some abuse had been discovered, was turned into a rule. A shell script was written (see program "countuser") to record multiple simultaneous logins from the same account, a second was written to terminate connections after two hours.

The CUUG CRC was established to assist companies in their open system business computing requirements, not substitute permanently for inadequate computing facilities or to actually run a business. Therefore, CUUG CRC use is limited to twenty hours per week (in person) and six hours per day (in two hour segments) by phone.

Companies with more than five accounts were encouraged to establish a UUCP feed to obtain their news and mail, rather than clogging the lines with individual access. Times were allocated and dedicated to these companies for UUCP access.

Anonymous FTP, FSF Software Distribution

CUUG purchased a full set of FSF tapes for gcc, emacs, and the X Window System, and distributed these materials throughout Southern Alberta. Members were encouraged to use the actual tapes when possible.

An anonymous ftp site called anonymous@cuug486.cuug.ab.ca was established for the exchange of newsletter articles and transcripts from open systems presentations between CUUG and other groups like CUUG around the world.

Non-Technical System Administration Issues

Many issues pertaining to public access Internet administration have no direct link with technology, but are equally problematic as the technical issues. Many of these "non-technical" issues cross over into the technical world when a business decision, such as "CUUG members may not share their accounts" results in having to write a program which enforces that decision.

Legal protection was the largest of the non-technical system issues. A CUUG CRC usage agreement was developed, and had to be signed and the original on file at the CUUG office to ensure that CUUG was not liable for any financial or business loss associated with work done on the CRC, and that the CRC was protected if CRC users were conducting illegal activities.

Other legal issues involved careful investigation of the somewhat contradictory rulings of the Royal Canadian Mounted Police (RCMP) and Canadian Radio and TeleCommunications (CRTC) with regards to some of materials distributed in the rec.* and alt.* news groups.

The CUUG CRC currently has a budget of approximately $21,000 CDN per year to cover the current level of services, which includes the costs of connecting to the Internet, telephone lines, equipment lease, and miscellaneous costs such as backup tapes. CUUG membership is $100 per year, with corporate discounts available. The remainder of the

the costs of operating the CRC come from CUUG corporate sponsorships.

Conclusions

The presence of the CUUG CRC has made a large impact in Calgary, especially in the area of small business. There have been the instantaneous benefits of access to e-mail, ftp, archie, gopher. Many companies utilizing the CUUG CRC have reported that they are now able to discuss problems and opportunities with a much larger group reflecting more international concerns than every before. Many people have used the CUUG CRC to teach themselves UNIX, or learn a new UNIX platform without cost in classes or hardware acquisitions. Small software companies have been able to port applications more easily, and everyone has been able to evaluate more combinations of hardware and software than they would before the CRC came into existence.

Internet access should be available to anyone who wants it at a reasonably affordable cost. A computer environment such as the CUUG CRC is one way of providing this kind of access.

Author Information

Sheri Byrne has a B.Sc. in Information System Management from the University of San Francisco. She has worked in the field of UNIX and open systems for the past 9 years, the last 4 at Gemini Learning Systems in Calgary, Alberta, Canada, where she is Director of Technical Operations. Sheri was a founding member of the Calgary UNIX Users Group, and was a member of the board for two terms. She can be reached by mail at 1750, 101-6th Avenue S.W. Calgary, AB, Canada, T2P 3P4, or electronically at byrne@posc.org until November 29, 1993, and byrnes@cuug.ab.ca after that.

APPENDIX

Intruders

```
# Make a log of all users who have tried to "su" another account.

olddate='grep "sun" intruders.out | cut -c1-15 | tail -1'
echo $olddate
date >> intruders.out
echo "---------------------------------------------" >> intruders.out
grep "su:" /usr/adm/messages | grep -v "byrnes" | grep -v "ingoldsb" >> intruders.out
echo "---------------------------------------------" >> intruders.out
echo " " >> intruders.out
echo " " >> intruders.out
```

Monitor

```
# This program is designed to monitor users logged into the system.
# This program is executed intentionaly as an endless loop. It checks
# for logged in users.  If no users are on the system, the process
# sleeps for about 30 seconds and then wakes up and checks the
# system for users again.  Once logins are detected, the users are
# monitored until they have all logged out and then repeats the
# sleep sequence.

while [ $-z $done ]
do
  list='users'
#
  if [ -z "$list" ]
  then
     sleep 30
#
  else
     echo "" >>userlogs
     w | grep -v 'sleep 300' >> userlogs
     echo "" >>userlogs
     sleep 30
  fi
#
```

```
# Writes the user information into a file called "userlog"
#
# A report is generated at midnight using all the information
# collected by the userlog. The userlog is then emptied for
# further use by the process.
# The name of the reports are generated by taking the system
# time and cuting the date from it (e.g. the name should look like
# this Jul17.rpt).

time='date | cut -c12-19'

if [ "$time" -eq "00:00:00" ]
  then
   month='date | cut -f2 -d" "'
   day1='date | cut -c9'
   day2='date | cut -c10'

   if [ "$day1" -eq " " ]
     then
       day1=0
   fi

   filename=$month$day1$day2.rpt
   cat userlogs >> $filename
   echo "">userlogs
fi
done
```

Deactivate

```
# Program runs before system login and to either call system login
# program or not.

# This program will accept user's login name, check to see if it exists
# in the file called 'deactive.tmp'.  This file contains the login
# names and renewal dates for selected users.  Appropriate messages as
# well as either allowing the login to proceed or being denied login
# permission occur in this program.

echo "Type your logname."
read logname
match1='grep $logname /users/cuug/deactive.tmp'
echo $match1
count=0
for name in $match1
do
   if [ "$name" = "$logname" ]
   then
      match=$name
      count=1
      echo "$match is match $count"
   elif [ "$count" -ge 1 -a "$count" -le 3 ]
   then
      match='echo $match $name'
      count='expr $count + 1'
      echo "$match is match $count."
   fi
done
echo "In deactive file as $match."

# Check to see if there is a match in the deactive.tmp file or not.
# If no match, let user login.  If there is a match, set variables
# and continue with validity checks.
```

```
if [ -z "$match" ]
then
#    sh /bin/login.unix
    echo "no match."
    exit
else
    deleteyear='echo $match | cut -f4 -d" "'
    deletemonth='echo $match | cut -f2 -d" "'
    deleteday='echo $match | cut -f3 -d" "'
    currentyear='date '+ %y''
    currentmonth='date '+ %m''
    today='date '+ %d''
fi

# If there is a match, send message and do appropriate action as
# described in the following program segment.

if [ "$deleteyear" -eq "$currentyear" ]
then
    if [ "$deletemonth" -eq "$currentmonth" ]
    then
        if [ "$deleteday" -lt "$today" ]
        then
            echo "Membership expired on $deletemonth/$deleteday/$deleteyear."
            echo "Contact Janet at 265-2289 for renewal."
            exit
        elif [ "$deleteday" -eq "$today" ]
        then
            echo "Your membership expires today.  Contact Janet at 265-2289"
            echo "for renewal."
#            sh /bin/login.unix
            exit
        elif [ "$deleteday" -gt "$today" ]
        then
            echo "Membership expires on $deletemonth/$deleteday/$deleteyear."
            echo "Contact Janet at 265-2289 for renewal before this date. "
#            sh /bin/login.unix
            exit
        fi
    elif [ "$deletemonth" -lt "$currentmonth" ]
    then
        echo "Membership expired on $deletemonth/$deleteday/$deleteyear."
        echo "Contact Janet at 265-2289 to reinstate. "
        exit
    else
        echo "Membership expires on $deletemonth/$deleteday/$deleteyear."
        echo "Contact Janet at 265-2289 for renewal before this date. "
#        sh /bin/login.uni
    fi
elif [ "$deleteyear" -gt "$currentyear" ]
then
    echo "Membership expires on $deletemonth/$deleteday/$deleteyear."
    echo "Contact Janet at 265-2289 for renewal before this date."
#    sh /bin/login.unix
else
    echo "Membership expired on $deletemonth/$deleteday/$deleteyear."
    echo "Contact Janet at 265-2289 to reinstate."
fi
exit
```

Starout

```
# Script to replace the "!" in the passwd file with a "*" to prevent a
# user from logging on. Also enters the user into the deleted.users file.

# find the user
cp /etc/passwd /etc/pass.sav
continue='y'
clear
echo " "
while [ "$continue" = 'y' ]
do
   echo " "
   echo "   Enter user to delete : "\c
   read user
   echo " "
   usrline='grep -n -i $user /etc/passwd'
   usr='echo $usrline | cut -f6 -d":"'
   lne='echo $usrline | cut -f1 -d":"'
   zap='N'
   if [ "$usr" ]
   then
      echo "       Remove user "\c
      echo $usr "(y/n)  "\c
      read zap
      echo " "
      if [ $zap = 'y' ]
      then
          sed """$lne s/!/*/""" /etc/passwd > /etc/delusrtemp
          mv /etc/delusrtemp /etc/passwd
      else
          echo "  Deletion Terminated !"
      fi
   else
      echo "       User not found ! , Press Enter "
      read jnk
   fi
   echo " "
   echo "   Continue Deletions ? (y/n) "\c
read continue
done
grep -n "\*" /etc/passwd | cut -f1,2,6 -d: > /etc/deleted.users
```

Corekiller

```
# shell program to find and remove all core files.

# core files are the saved images of programs that crash. these files
# can be extremely large so it is necessary to periodically seek out
# and destroy them.

coreusr='find / -name core -print -exec rm -f {} \; | \
                     cut -f3 -d"/" | tee cores.usr'
echo $coreusr

for cusr in $coreusr
do
      qusr='grep $cusr /etc/passwd | cut -f1 -d":"'
      if [ "$qusr" = "$cusr" ]
      then
            mail $cusr < cleanup.mes
      fi
done
```

Delete_users

```
# This file is run two months after CUUG membership expiration to remove
# obsolete accounts

cp /etc/passwd foo

tail +13 foo | grep ":\*:" | grep -v fsf | cut -f6 -d":" > dirs_to_hose

for file in 'cat dirs_to_hose'
do
   echo $file
   echo -n "Do you want to delete this guy?"
   read answer
   if [ "$answer" = "y" ]
   then
      rm -rf $file
      echo $file deleted
   else
      echo $file NOT deleted
   fi
done
```

Countuser

```
# Count how many times each user is currently logged into system and, if more
# than one, send user and root messages via electronic mail to inform them of
# time, date and number of logins.

howmany='who|wc -l'
noduplic='who|cut -c1-8|sort -u|wc -l'
if [ "$howmany" -eq "$noduplic" ]
then
    exit
else
    unset howmany
    unset noduplic
    howmany='who|cut -c1-8'
    noduplic='who|cut -c1-8|sort -u'
    for activename in $noduplic
    do
        count=0
        for name in $howmany
        do
            if [ "$name" = "$activename" ]
            then
                count='expr $count + 1'
            fi
        done
        if [ "$count" -gt 1 ]
        then
            echo "'date|cut -c1-10'">badlogin
            echo $activename>>badlogin
            echo "Currently have $count logins">>badlogin
            echo "More than one Login. Contact CUUG.">>badlogin
            mail $activename root<badlogin
            rm badlogin
        fi
    done
fi
```

Disk_pig

```
# performs a comparative analysis of previous disk usage vs current disk
# usage to find out who is being a "pig"

now='date | cut -c5-7'
before='ls -l pig2.new | cut -c33-35'
rightnow='date | cut -c9-10'
earlier='ls -l pig2.new | cut -c37-38'
withinweek='expr $earlier + 7'
home_pct='df | grep home | cut -c48-50'
if [ "$now" = "$before" ]
then
   if [ "$home_pct" -lt "91" -a "$rightnow" -lt "$withinweek" ]
   then
      exit
   fi
fi

sort -u -o temp2.new pig2.new
sort -u -o temp2.old pig2.old
rm newpigs.new
echo "'date | cut -c1-10': Last months new overusers are : ">newpigs.new
comm -13 temp2.old temp2.new>>newpigs.new
rm -f temp2.old temp2.new pig2.old
cp pig2.new pig2.old
rm pig2.new
echo "">pig2.new
        echo "$user doesn't exist in current directory." >> /home/coop/pig2.new
directories='egrep '(100|150|200)' /etc/passwd | cut -f6 -d":"'
for user in $directories
do
   if [ -d "$user" ]
   then
      cd $user
      amt='du -s | cut -f1 -d" "'
      if [ "$amt" -gt "1000" ]
      then
                  echo $user uses $amt
                  echo $user uses $amt >> /home/coop/pig2.new
      fi
   else
      echo " $user does not exist in the current directory structure"
   fi
done
```

Hogmail

```
# Send mail to all the users who are hogging disk space.

filen="pig2.new"
food="dskmes"
date >> hogmail.out
hogs='cat $filen | cut -f3 -d"/" | cut -f1 -d" "'

for pig in $hogs
do
      if [ "$pig" != "byrnes" -a "$pig" != "empress" ]
        then
            mail $pig < $food
            echo $pig is a pig >> hogmail.out
      fi
done
```

Implementing Execution Controls in Unix

Todd Gamble – WilTel Network Services

ABSTRACT

Current implementations of UNIX offer security features in the form of discretionary access controls (DACs). DACs are implemented with file access permissions and access control lists (ACLs). Unfortunately, neither of these facilities provide for access control to active processes. In order to provide many users access to a process (and its associated data) the current practice at our site is to establish a group account, where members on a project team share the login and password for an application. This practice is both insecure [cur90][fer93], and a violation of our site's security policies.

This paper describes the implementation of a new tool, `medex`, which eliminates the need for group login accounts. `Medex` mediates the access of users to privileged accounts and executables. The history behind our use of group accounts and a complete methodology for UNIX application management are presented. Details of the implementation of `medex`, including its interaction with the existing security features of UNIX, are given. The tool utilizes execution control lists (ECLs) as a means to allow controlled execution of programs under accounts other than the current login. `Medex` also re-authenticates the user's password upon each instantiation and maintains an audit trail via log files or the use of the UNIX *syslog* facility. A complete project management example utilizing `medex` is given along with a comparison to related tools.

Site Description

WilTel is the forth largest telecommunications company in the U.S. The company's nationwide enterprise network has an installed base of approximately 200 UNIX platforms, including UNIX variants from DEC, HP, IBM, NeXT, SCO, Sequent, and Sun. The user base is in the thousands with the largest system supporting approximately 600 accounts. Typical UNIX application projects include distributed databases, device control and data acquisition, network monitoring, and customer access.

History

Many projects developed at WilTel have inherited support systems which rely upon the use of group accounts. Project teams share the password to a single login account that is used by all team members to manage an application. Team members login or *su*(1) to a special account that is used to start up the application or to modify data owned by the application. While logins or *su*(1) usage can be tracked via conventional accounting records on the machines, the commands executed by the user once in the account are limited only by the access authorizations for the special account, i.e., not those of the specific user. Delegation of authority has little granularity; either a user has the password to the special account or not. Propagation of the password from user to user is untraceable. Modification of the password (e.g., aging) is difficult since password changes must be scheduled with all users of the special account.

Another common practice is to establish "check out" logins. A special account is created that can perform some privileged function. Users "check out" the account by obtaining its current password. After each use, the password is changed. This eliminates the problem of password propagation, but still does not provide a means to limit the user's actions once access to the special account is given.

Requirements

Our objective is to eliminate the need for *group accounts*. They are difficult to maintain and are insecure. However, we still need the ability to define a *privileged* set of users who are capable of accessing an application project's data as well as its processes during execution. Additionally, the number of access points to the project data should be controlled while providing a complete audit trail of all accesses.

A separation of duties should be established such that authorizations to the system may be made by *security administrators* while system maintenance is performed by *system administrators*. The system should enable security administrators to define strict controls on precise commands that could be executed by a specific user with the account privileges of another user. The system should be portable across all the UNIX environments. Finally, any tools developed should be integrated with the vendor's existing security system.

Access Controls

Most of the UNIX versions at our site implement access controls using industry standard Discretionary Access Controls (DACs).

DACs are defined as follows:

The [system] shall define and control access between named users and named objects (e.g., files and programs) in the ... system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals or defined groups or both. [dod85]

UNIX offers DACs in the form of file permission bits, i.e., the self/group/public controls. The permission bit field is located within the file's *inode.* An inode is a data structure that defines a file within the UNIX hierarchical file system. The permission bit field (see Figure 1) includes access flags for three classes of users: user (self), group, and other (public). Read, write, and execute permissions (or *modes*) for a file are specified by turning on (or off) bits within the bit field.

User			Group			Other		
read	write	exec	read	write	exec	read	write	exec
8	7	6	5	4	3	2	1	0

Figure 1: UNIX file permission bit field

Some UNIX implementations offer additional DAC mechanisms in the form of Access Control Lists (ACLs)[hew91]. ACLs offer more selectivity in access control than the standard file permissions. ACLs allow the file owner or superuser to permit or deny access to a list of users, groups, or combinations thereof. ACLs are supported as a superset of the UNIX operating system DAC mechanism for files, but not for other objects such as inter-process communication (IPC) objects.

An ACL consists of sets of (*user.group,mode*) entries associated with a file that specify permissions. Each entry specifies a set of access permissions for one *user.group* pair, including read, write, and execute (or search). In an ACL, user and group IDs can be represented by their mnemonic user and group names or by their numeric user ID number (UID) or group ID number (GID) as found in the `/etc/passwd` file. Two special symbols may also be used in the ACL entry to simplify the lists:

```
% – no specific user or group
@ – the current file owner or group
```

While DACs control access to file objects, no provision is made for objects of other types (e.g., active processes). Also, all access under the DAC model is centered around the user (or group). The level of granularity on controls does not extend to specifying which programs may be used to access a file object.

Execution Controls

Bacic [bac89] has proposed the Process Execution Control (PEC) model as a means to extend the security system in modern operating systems. PECs introduce additional steps in security by restricting the list of executables that have access to a file and by restricting which user can invoke which executable. These two restrictions are designed to preserve the *integrity* of system objects. Integrity is maintained if objects are manipulated only by *authorized* and *known* entities that leave the object in a *consistent* state after manipulation.

Using PECs, each object in the system is tagged with an Execution Control List (ECL), that describes *which users* may read/write an object and *which program* must be invoked to do so. This hides the object from the user while still allowing the user to manipulate the file with designated programs.

PECs view all files on a given system as *objects* whether or not they are executable. An *executable* is a binary encoded file that may be passed to the operating system for execution. A *subject* is the combination of a known system user and an executable that is accessible to the user. Objects are manipulated by subjects and subjects may in turn be manipulated by other subjects, making them an object from the new subject's viewpoint. Every object in the system defines a *domain* of control around itself. A domain is described by means of the ECL specifying which subjects may manipulate an object. An ECL entry is represented as a *<user,program,data>* triple authorizing a specific user to access a data object with a designated program.

PECs have been implemented in a version of Tunis[1] [bac90]. The kernel of Tunis was modified to consult ECLs when system calls are made for file access. The lists consist of *<user,program>* pairs that are attached to file objects via their inode in a fashion similar to ACLs. These pairs define the subjects that may access the object. The Tunis system kernel functions as the mediation point between subjects and file objects by consulting the ECL attached to the file's inode whenever a system call (e.g., *write*) is made to access the file.

The PEC model extends controls to restrict access to data to pre-defined user and program pairs, but does not include a specification for access control to processes.

[1]Tunis is a UNIX compatible operating system developed at the University of Toronto.

Medex Design

The existing DACs in UNIX and the PEC model provide most of the necessary foundation to develop a system which satisfies the original requirements. Since modification of the UNIX kernel as done for implementation in Tunis is not an option at our site, a solution based on a combination of an application program and careful configuration of the available control mechanisms is used. The application program is implemented as `medex`, an execution control facility that mediates access between users and privileged objects. The system configuration requirements are the implementation of a control policy for project resources on the system.

object: Either a file on the system or the invocation of an executable file as an active process.
method: An executable that is authorized to access a particular object, i.e., the executable is a *method* of the object[2].
subject: The combination of a known system user and a method that is accessible to the user.

Figure 2: Definitions for `medex` control facility

Figure 2 defines the `medex` control elements. Users on the system are authorized to access objects via defined methods. With proper configuration, the superuser is the only user able to subvert the controls imposed by the `medex` facility.

Normally, the objects of concern are those which are part of a particular application project. A *project* can be defined as the *class* of all objects that represent parts of the project on the system. In order to identify objects within a class, a project account is created on the system. The UID of the project account is used as a class identifier for the project's objects. All file objects within a project class should be owned by the UID of the project account. In addition, the project account should disallow direct logins, i.e., its password should be disabled. This eliminates access to the project's objects except for privileged intermediaries (e.g., `medex`), thus, hiding the objects from the user.

Methods are specified in the control file `medex.methods`. Entries in the file associate a UNIX executable with the UID of the project account. In order to be a method of a project object the executable must execute under this UID. Each entry is tagged with the method name that will be used to authorize access to the method (see Figure 3). Method names need not be related to the actual executable (i.e. they can be used to provide a some level of abstraction), but they must be unique. Executables are specified as UNIX pathnames followed by any command line options. The pathnames must be *absolute*, i.e., specified from the root of the directory hierarchy with a leading "/". This is done in order to reduce the possibility of a trojan horse being introduced in the execution path of the `medex` program.

```
METHOD-NAME:uid,pathname [options]
```

Figure 3: `medex.methods` file

Users are granted access to defined methods via authorizations in the `medex.ecl` file. Each line in the file represents an ECL entry that associates a *<uname.gname>* pair with a method. A user with a username of *uname* and group membership in *gname*[3] may invoke the executable defined in the method with the method's UID, i.e., the user may act as the project account to run the program given in the method's specification. For example if methods are defined as in Figure 4 and ECLs as in Figure 5 then user *jane* in group *programmer* can access objects in the project *bigapp* with the methods PRG1 and GUITOOL.

```
PRG1:bigapp,/usr/proj/bigapp/prog1
GUITOOL:bigapp,/local/guis/tool1
```

Figure 4: Example `medex.methods` file

```
jane.programmer:PRG1, GUITOOL
```

Figure 5: Example `medex.ecl` file

The ECL file syntax includes the special character `%` with a similar meaning as in ACLs. The percent sign "`%`" is used to indicate a wild card entry, i.e., any user or group. For example, with the ECL file in Figure 6, **all** users in the group *programmer* are given access to the method GUITOOL, while *jane* is the only user that may access the PRG1 method. Note that *jane* can still access GUITOOL since she is a member of the *programmer* group.

```
jane.programmer:PRG1
%.programmer:GUITOOL
```

Figure 6: Example `medex.ecl` revised file

Medex Implementation

`Medex` is implemented in Perl [wal91] for compatibility with all the required UNIX variants. The special features of Perl for analysis of tainted execution paths are used in order to minimize the possibility of trojan horses or other programmed threats based on executable imposters. Both the ECLs and the method specifications are stored in protected files owned by the superuser account.

Methods are stored internally using a Perl associative array which is keyed on the method name.

[2]This definition of *method* has similar semantics to the methods defined in Smalltalk [gol83].

[3]Group memberships are defined in the file `/etc/group` on most UNIX systems.

ECL entries are stored as an array of regular expressions. When a request is received by an invocation of medex, the requester's *<uname.gname>* pair and the method name given on the medex command line are compared to the ECL. If a match is found then the method is executed for the requester and a log entry is generated either to the file medex.log or by issuing a call to UNIX *syslog* facility. On each invocation medex queries the user for her login password in order to verify her identity. This eliminates the possibility of an unknown user accessing a privileged account via an abandoned login. Note in Figure 7 that the user *jane* calls medex by specifying its absolute pathname, a good idea when executing any setuid program.

```
jane% /usr/local/bin/medex PRG1
Password: xyz123
```

Figure 7: Example medex invocation

The medex program must operate as the privileged user root in order to switch its UID from the requesting user to that of the method's project. Care must be taken to create methods which do not allow the user to escape to a UNIX command shell and gain additional access not defined in the method. Therefore, method declarations for UNIX utilities that allow shell escapes (e.g., *vi*) are highly discouraged.

The reader will note that *user* and *group* names, instead of numeric UIDs and GIDs, were used in the medex control file examples above. In fact, the use of numeric UIDs and GIDs is not supported. Experience has shown that these are much less stable than the specific user and group names that are assigned. Since only the names are used in the medex files, the problems with UID and GID migration are avoided. Also, references to the user's *group* membership work for **all** groups in which the user is a member, not just his or her current one[4]. This eliminates the need for the user to issue a *newgrp* call in order to switch groups prior to invoking medex.

Medex Usage

The procedures for medex usage include the establishment of special accounts with disabled logins, creation of groups for application development or support teams, and delegation of access authority to special accounts via the ECL control file. This procedure provides a mechanism for enforcing access via medex to the special account.

Further implementation of PECs can be accomplished by changing the file permissions on executables and data to make them available only to the special account, thereby limiting the paths by which the data can be accessed. Management of the ECL file can be delegated to a security administrator for proper separation of duties by changing the file's access permissions to make the file writable by the security administrator. This can also be accomplished with the use of an ACL on the file, specifying access for the administrator's account.

Example 1: Application Project Management

In this section we present a small example to illustrate the use of medex when managing a database application. We often have the need to establish processes which collect data in real-time from devices and store the data in a database. Since the application is updating the database, it must operate under the login of a valid database user. Also, the application must be supported by a rotating support staff in order to provide for 24 hour data collection from the device.

To provide for this type of access, we first establish a login account for the application. For this example, we choose the account name *rtapp* to match the application's name, i.e., rt. The account *rtapp* is also given write access to the necessary database files on the database server. We disable logins on the *rtapp* account by placing an "*" in the password field of the account's line in the /etc/passwd file.

Two directory areas are created on the system, one, /proj/rt/dev is a development area, and the other, /proj/rt is a production area. All files in the production area are given the owner *rtapp*, essentially putting all production access under medex control.

Once the special account is enabled, we establish methods to control the application in the medex.methods file as in Figure 8. Methods are established to start the application, stop the application, reload a configuration file, and to update the production code from a development area. The RTSTOP method is implemented as shell script that issues a KILL signal to rt. The RTUPDATE method is a program that migrates development code from the directory /proj/rt/dev to the production directory.

```
RTSTART:rtapp,/proj/rt/init
RTSTOP:rtapp,/proj/rt/kill
RTRECON:rtapp,/proj/rt/config
RTUPDATE:rtapp,/proj/rt/update
```

Figure 8: medex.methods file for rt

We establish the group *rtappops* in the /etc/group file; filling the group with the usernames of the support personnel for the application. Authorizations for members of the group to access the *rtapp* methods are defined in the medex.ecl file as in Figure 9. All users in the group *rtappops*

[4]This is primarily an issue in UNIX System V where users may only belong to one group at a time.

are allowed to start and stop the application. Only the user *sally* is allowed to reconfigure the application. The user *tim*, the application department's code librarian, is allowed to move code from development to production. Note that it is not necessary for *tim* to be a member of the group *rtappops*.

```
%.rtappops:RTSTART,RTSTOP
sally.rtappops:RTRECON
tim.%:RTUPDATE
```

Figure 9: Changes to `medex.ecl` file for `rt`

With this configuration, all transactions to the application `rt` are logged with audit trail information indicating the specific user that modified or accessed the application. Code migration from development to production is also controlled and logged, making revision and quality control easier.

Example 2: Running DNS

In this section we present an example of using `medex` to control access to a pre-existing account, namely the `root` account. We operate the DNS (Domain Name System) at our site using the Berkeley Internet Name Domain (BIND) software. The system implements the program *named*(8) as a UNIX daemon which processes queries for the DNS database. The daemon is usually started by *root* from one of the system startup files. Since the daemon is constantly running, it accepts commands via *signal*(3) calls, rather than from the command line. Under normal operation a UNIX process will only accept signals from its owner, in this case *root*. For all system administrators or operators that we wish to have access to `named`, we must give them access to the *root* account. Rather than just hand out the password to *root's* account, we use `medex` to enable *root* access for the limited set of executables necessary to control `named`.

When `named` starts, it reads the current DNS database from a set of files called *zone* files. These files contain the necessary data for the DNS server to reply to queries about hostname to address mapping information. Since the syntax for DNS zone files is somewhat tricky, we automatically generate our files from a host table that is in the same format as the standard UNIX `/etc/hosts` file. The translation is performed by a tool called `h2n`[5] (hosts to named).

Once `named` is running we need our BIND operators to be able to:

- Update the `hosts` file and rebuild the DNS databases on the server with `h2n`.
- Issue a HUP signal to `named` telling it to reload the databases into its active memory.
- Restart `named` by issuing a KILL signal and running the server daemon by running `/etc/named` as *root*.

First we create a new group in the `/etc/group` file by adding a line like:

```
dnsops:*:100:charles,tony,david
```

This sets up the group `dnsops` with group id of 100 and puts our DNS operators (Charles, Tony, and David) in the group. We use the new group to authorize access the DNS methods. We add the lines in Figure 10 to `medex.methods` to support the required commands. The shell script `reload` issues a HUP signal to `named`. The file `restart` is a shell script that sends a KILL signal to `named` and then re-executes `named`. The use of these scripts illustrates using `medex` to control access to an active process, i.e., the `named` daemon.

```
DNSRELOAD:root,/var/named/reload
H2N:root,/var/named/h2n
DNSRESTART:root,/var/named/restart
```

Figure 10: Changes to `medex.methods` file

```
%.dnsops:DNSRELOAD,DNSRESTART,H2N
```

Figure 11: Changes to `medex.ecl` file

With the commands setup we can authorize the DNS operators with one line in `medex.ecl` (see Figure 11). Though this procedure implements all the necessary execution controls, the DNS operators still need to change the `hosts` file. This is done by making the file's group `dnsops` and enabling group write permission on the file. We also put the file under revision control using the revision control system (RCS) so that changes are tagged with the specific username of the DNS operator.

Comparison to Other Tools

Two tools exist with similar functionality to `medex`. One, `sudo`[6], allows controlled access to the superuser account. Commands (and command groups) may be aliased and access to them authorized in a control file called `sudoers`. Sudo satisfies many of the requirements of `medex`, such as a protected access mechanism to the privileged account and an audit trail, however it is limited to access of the superuser account. It includes a more complex syntax that is accessed with a parser written in `lex` and `yacc`. The remainder of the program is written in C. These features made it less portable than a utility developed in Perl.

A second tool, `su-someone`[7], allows a specified group of users to switch to a special user's

[5]`h2n` is presented by Albitz and Liu in their book *DNS and BIND* from O'Reilly and Associates. I highly recommend it for anyone considering running a DNS server.

[6]`Sudo`'s current incarnation was developed by Jeff Nieusma and David Hieb at The Root Group, Inc.

account. This stops the password to the special account from being propagated, but it doesn't restrict the actions of the user once they have made the switch. The access list is compiled into the program, i.e., a new executable is created for each application of the tool.

Conclusions

During the implementation of `medex` it became glaringly apparent that most of the work involved developing a model for application project management, rather than actually implementing a tool to provide a controlled access to privileged accounts. Along with the implementation were other procedural issues, such as procedures for setting up special accounts and for establishing adequate separation of duties. Questions raised in this area may lead to future extensions that allow delegation of method and ECL specifications for a particular project class to a project security administrator.

Since some of the ECL specification syntax relies on the use of group assignments in the `/etc/group` file, there is a possible limitation on the number of methods and projects that a user may access. This limitation is imposed by the operating system restricting the number of concurrent groups that a user may belong to at one time. An additional group membership file could be added, but the use of existing DACs in the form of the `/etc/group` was a desirable feature.

Availability

Release information for `medex` is available via anonymous FTP from `ftp.wiltel.com` in the directory `/pub/src/medex`. See the `README` file in this directory.

Author Information

After receiving his M.S. in Computer Science in 1991 from Washington University in St. Louis, Todd Gamble worked as a UNIX system administrator for the university until joining WilTel in July of 1992. He is currently project lead of Network Security for WilTel. His recent project areas include network firewalls, Internet customer access, and Unix security. He can be reached via electronic mail at the address `todd_gamble@wiltel.com`.

References

[bac89] Eugene Mate Bacic, "Process Execution Controls as a Method to Ensure Consistency", Fifth Annual Computer Security Applications Conference, pp. 114—120. 1989.

[bac90] Eugene Mate Bacic, "Process Execution Controls: Revisited", Canadian System Security Centre, Communications Security Establishment, 1990.

[cur90] David A. Curry, "Improving the Security of Your UNIX System", SRI International Publication, ITSTD-721-FR-90-21, April 1990.

[dod85] Department of Defense, *Trusted Computer System Evaluation Criteria,* DOD 5200.28-STD, National Computer Security Center, Fort Meade, Maryland, December 1985.

[fer93] David Ferbrache and Gavin Shearer, *UNIX Installation Security & Integrity*, Prentice Hall, Englewood Cliffs, 1993.

[gol83] Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, May 1983.

[hew91] *HP-UX Release 8.05 System Manuals*, Hewlett-Packard Company, June 1991.

[wal91] Larry Wall and Randal L. Schwartz, *Programming Perl*, O'Reilly and Associates, Sebastopol, CA, 1991.

[7] `su-someone` was developed by Wietse Venema at the Eindhoven University of Technology.

HLFSD: Delivering Email to Your $HOME

Erez Zadok – Columbia University
Alexander Dupuy – System Management ARTS

ABSTRACT

We consider the problem of enabling users to access their mailbox files from any host on our local network, and not only on one designated "home machine". We require a solution which will not introduce any new single points of failure, force us to modify mail transfer agents and user agents, or require changes to the operating system kernels. In other words, minimize the amount of work needed by system-administrators and users. Our solution is to deliver mail into the users' home directories, which are exported via NFS[20,25] to all of the machines on our network. We wrote a small user-level NFS server implementing a single symbolic link that references the home directory of a user, either the one who accessed it, or by name, with a fallback reference in case of failures. This enables electronic mail to be delivered directly into the user's home directory, which is already accessible from any machine on the network. Although we have used our server primarily for mail delivery redirection, it can be used to redirect spooled faxes, access to `/tmp`, etc.

Introduction

Electronic mail has become one of the major applications for many computer networks, and use of this service is expected to increase over time, as networks proliferate and become faster. Providing a convenient environment for users to read, compose, and send electronic mail has become a requirement for systems administrators (SAs).

Widely used methods for handling mail usually require users to be logged into a designated "home" machine, where their mailbox files reside. Only on that one machine can they read newly arrived mail. Since users have to be logged into that system to read their mail, they often find it convenient to run all of their other processes on that system as well, including memory and CPU-intensive jobs. For example, in our department, we have allocated and configured several multi-processor servers to handle such demanding CPU/memory applications, but these were under-utilized, in large part due to the inconvenience of not being able to read mail on those machines. (No home directories were located on these designated CPU-servers, since we did not want NFS service for users' home directories to have to compete with intensive jobs. At the same time, we discouraged users from running demanding applications on their home machines.)

Many different solutions have been proposed to allow users to read their mail on any host. However, all of these solutions fail in one or more of several ways:

- They introduce new single points of failure
- They require using different mail transfer agents (MTAs)[15] or user agents (UAs)
- They do not solve the problem for all cases, i.e., the solution is only partially successful for a particular environment.

We have designed a simple filesystem, called the *Home-Link File System*, to provide the ability to deliver mail to users' home directories, without modification to mail-related applications. We have endeavored to make it as stable as possible. Of great importance to us was to make sure the HLFS daemon, `hlfsd`, would not hang under any circumstances, and would take the next-best action when faced with problems. Compared to alternative methods, `hlfsd` is a stable, more general solution, and easier to install/use. In fact, in some ways, we have even managed to improve the reliability and security of mail service.

Our server implements a small filesystem containing a symbolic link to a subdirectory of the invoking user's home directory, and named symbolic links to users' mailbox files. An example, using pathnames from our environment, is depicted in Figure 1.[1]

The `hlfsd` server finds out the *uid*[2] of the process that is accessing its mount point, and resolves the pathname component home as a symbolic link to a subdirectory within the home directory given by the *uid*'s entry in the password file (see Table 1). If the *gid* of the process that attempts to access a mailbox file is a special one (called HLFS_GID), then the server maps the name of the next pathname

[1] In Figure 1, ~*NAME* is the home directory of the user whose user-name is *NAME*; ~*USER* is the home directory of the user with user-id *uid*.
[2] NFS uses effective *uids*.

component directly to the user's mailbox (Table 2). This is necessary so that access to a mailbox file by users other than the owner can succeed. The server has safety features in case of failures such as hung filesystems or home directory filesystems that are inaccessible or full.

On most of our machines, mail gets delivered to the directory /var/spool/mail.[3] Many programs, including UAs, depend on that path. Hlfsd creates a directory /mail, and mounts itself on top of that directory. Hlfsd implements the path name component called home, pointing to a subdirectory of the user's home directory. We made a symbolic link from /var/spool/mail to /mail/home, so that accessing /var/spool/mail actually causes access to a subdirectory within a user's home directory.

The rest of this paper is organized as follows. The second section discusses in detail the problems and limitations of other home-mail-delivery methods. The third section details the design of the *Home-Link File System* and the fourth section describes the implementation of hlfsd. The next section

[3]Other directories used for this purpose are /var/mail on SVR4, /usr/mail on other System V-based operating-systems, and /usr/spool/mail on BSD-based systems.

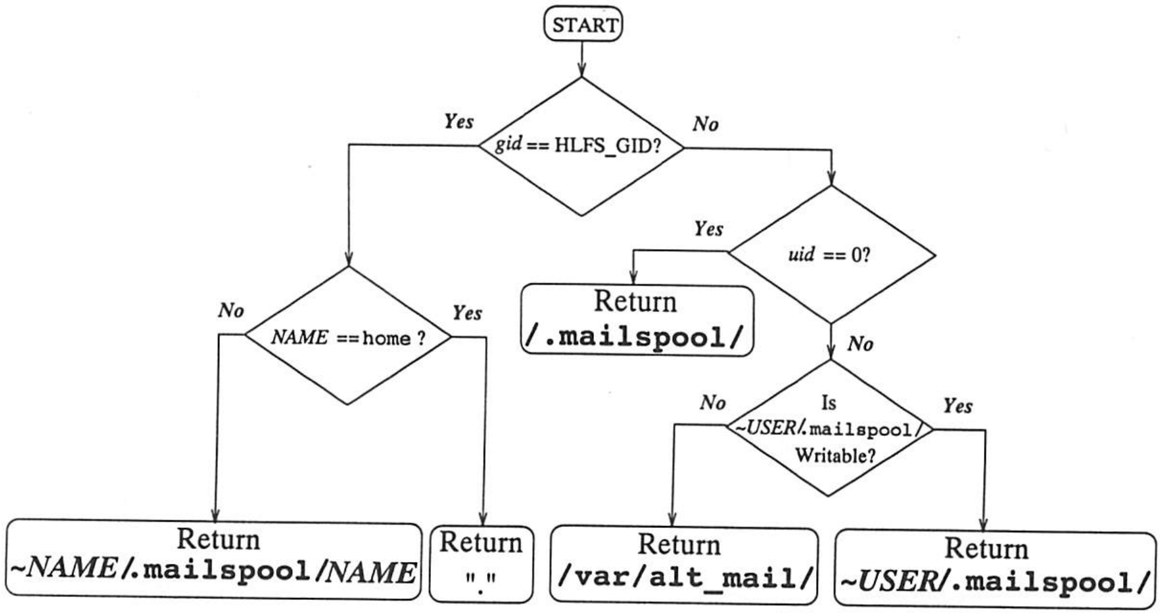

Figure 1: Hlfsd resolving the *NAME* component of /mail/*NAME*

Conditions: *uid* =ezk, *gid*≠HLFS_GID, and /users/ezk/.mailspool/ is writable.

Resolving component	Pathname left	Value if symbolic link
/	var/mail/*NAME*	
var/	mail/*NAME*	
mail@	/mail/home/*NAME*	mail@ ⇒ /mail/home
/	mail/home/*NAME*	
mail/	home/*NAME*	
home@	*NAME*	home@ ⇒ /users/ezk/.mailspool
/	users/ezk/.mailspool/*NAME*	
users/	ezk/.mailspool/*NAME*	
ezk/	.mailspool/*NAME*	
.mailspool/	*NAME*	
NAME		

Table 1: Resolving /var/mail/*NAME* to /users/ezk/.mailspool/*NAME*

evaluates our implementation. Related systems, conclusions, future directions, and alternative uses are described in the final two sections.

Background

This section provides an in-depth discussion of why available methods for delivering mail to home directories are not as good as hlfsd's method.

Single Host Mail Spool Directory

The most common method for mail delivery is for mail to be appended to a mailbox file in a standard spool directory on the designated "mail home" machine of the user. The greatest advantage of this method is that it is the default method most vendors provide with their systems, thus very little (if any) configuration is required on the SA's part. All they need to set up are mail aliases directing mail to the host on which the user's mailbox file is assigned. (Otherwise, mail is delivered locally, and users find mailboxes on many machines.)

As users become more sophisticated, and aided by windowing systems, they find themselves logging in on multiple hosts at once, performing several tasks concurrently. They ask to be able to read their mail on any host on the network, not just the one designated as their "mail home."

Centralized Mail Spool Directory

A popular method for providing mail readability from any host is to have all mail delivered to a mail spool directory on a designated "mail-server" which is exported via NFS to all of the hosts on the network. Configuring such a system is relatively easy. On most systems, the bulk of the work is a one-time addition to one or two configuration files in /etc. The file-server's spool directory is then hard-mounted across every machine on the local network. In small environments with only a handful of hosts this can be an acceptable solution. In our department, with a couple of hundred active hosts and thousands of mail messages processed daily, this was deemed completely unacceptable, as it introduced several types of problems:

- **Scalability and Performance**: as more and more machines get added to the network, more mail traffic has to go over NFS to and from the mail-server. Users like to run mail-watchers[2,7] and read their mail often. The stress on the shared infrastructure increases with every user and host added; loads on the mail server would most certainly be high since all mail delivery goes through that one machine.[4] This leads to lower reliability and performance. To reduce the number of concurrent connections between clients and the server host, some SAs have resorted to automounting the mail-spool directory. But this solution only makes things worse: since users often run mail watchers, and many popular applications such as trn, emacs, csh or ksh check periodically for new mail, the automounted directory would be effectively permanently mounted. If it gets unmounted automatically by the automounter program[3], it is most likely to get mounted shortly afterwards, consuming more I/O resources by the constant cycle of mount and umount calls.
- **Reliability**: the mail-server host and its network connectivity must be very reliable. Worse, since the spool directory has to be hard-mounted,[5] many processes which access the spool directory (various shells, login, emacs, etc.) would be hung as long as connectivity to the mail-server is severed. To improve reliability, SAs may choose to backup the mail-server's spool partition several times a day. This may make things worse since reading or delivering mail while backups are in progress may cause backups to be inconsistent; more backups consume more backup-media resources, and increase the load on the mail-server host.

Distributed Mail Spool Service

Despite the existence of a few systems that support delivery to users' home directories,[6] mail delivery to home directories hasn't caught on. We believe the main reason is that there are too many programs that "know" where mailbox files reside. Besides the obvious (the delivery program /bin/mail and mail readers like /usr/ucb/Mail, mush, mm, etc.), other programs that know mailbox location are login, from, almost every shell, xbiff, xmailbox, and even some programs not directly related to mail, such as emacs and trn. Although some of these programs can be configured to look in different directories with the use of environment variables and other resources, many of them cannot. The overall porting work is significant.

Other methods that have yet to catch on require the use of a special mail-reading server, such as IMAP[16] or POP[17]. The main disadvantage of these systems is that UAs need to be modified to use these services – a long and involved task. That is

[4]Delivery via NFS-mounted filesystems may require usage of rpc.lockd and rpc.statd to provide distributed file-locking, both of which are widely regarded as unstable and unreliable. Furthermore, this will degrade performance, as local processes as well as remote nfsd processes are kept busy.

[5]No SA in their right minds would soft-mount read/write partitions – the chances for data loss are too great.

[6]AIX 1.2's bellmail for the IBM PS/2s[9], /bin/mail on SunOS for the Sun 386i machines, and zmailer[27].

why they are not popular at this time. See the section on mail-reading servers for more details.

Several other ideas have been proposed and even used in various environments. None of them is robust. They are mostly very specialized, inflexible, and do not extend to the general case. Some potentially lose or corrupt mail:

- **Automounters**: using an automounter such as amd[13] to provide a set of symbolic links from the normal spool directory to user home directories is not sufficient. UAs rename, unlink, and recreate the mailbox as a regular file, therefore it must be a real file, not a symbolic link. Furthermore, it must reside in a real directory which is writable by the UAs and MTAs. This method may also require populating `/var/spool/mail` with symbolic links and making sure they are updated. Making amd manage that directory directly fails, since many various lock files need to be managed as well (see the section on lock files). Also, amd does not provide all of the NFS operations which are required to write mail such as *write*, *create*, *remove*, and *unlink*.
- **$MAIL**: setting this variable to an automounted directory on the user's mail spool host only solves the problem for those programs which know and use $MAIL. Many programs don't, therefore this solution is partial and of limited flexibility. Also, it requires the SAs or the users to set it themselves – an added level of inconvenience and possible failures.
- **/bin/mail**: using a different mail delivery agent could be the solution. One such example is `hdmail`[6]. However, `hdmail` still requires modifying all UAs, the MTA's configuration, installing new daemons, and changing login scripts. This makes the system less upgradable or compatible with others, and adds one more complicated system for SAs to deal with. It is not a complete solution because it still requires each user have their $MAIL variable setup correctly, and that every program use this variable.

Why Deliver Into The Home Directory?

There are several major reasons why SAs might want to deliver mail directly into the users' home directories:

- **Location**: many mail readers move mail from the spool directory to the user's home directory. It speeds up this operation if the two are on the same filesystem. If for some reason the user's home directory is inaccessible, it isn't that useful to be able to read mail, since there is no place to move it to.[7] In some cases, trying to move mail to a non-existent or hung filesystem may result in mail loss.
- **Distribution**: having all mail spool directories spread among the many more filesystems minimizes the chances that complete environments will grind to a halt when a single server is down. It does increase the chance that there will be someone who is not able to read their mail when a machine is down, but that is usually preferred to having no one be able to read their mail because a centralized mail server is down. The problem of losing some mail due to the (presumably) higher chances that a user's machine is down is minimized in HLFS as described in the sections on alternate mail spool directories and avoiding hangs.
- **Security**: delivering mail to users' home directories has another advantage – enhanced security and privacy. Since a shared system mail spool directory has to be world-readable and searchable,[8] any user can see whether other users have mail, when they last received new mail, or when they last read their mail. Programs such as `finger` display this information, which some consider an infringement of privacy. While it is possible to disable this feature of `finger` so that remote users cannot see a mailbox file's status, this doesn't prevent local users from getting the information. Furthermore, there are more programs which make use of this information. In shared environments, disabling such programs has to be done on a system-wide basis, but with mail delivered to users' home directories, users less concerned with privacy who *do* want to let others know when they last received or read mail can easily do so using file protection bits. Lastly, on systems that do not export their NFS filesystem with `anon=0`, superusers are less likely to snoop around others' mail, as they become "nobodies" across NFS.

In summary, delivering mail to home directories provides users the functionality sought, and also avoids most of the problems discussed in the section that discusses the centralized mail spool directory.

Design

We named our file system the *Home-Link File System*, because that's all it does – provide symbolic

[7]This assumes that they can login to a different host. Some systems, such as HP-UX, do not allow login if they cannot `chdir` to the user's home directory.

[8]System V has the mail spool directory only group writable but that makes it more difficult to install other UAs or MTAs.

links to various files and directories in a user's home directory. The soft link has a fixed name, but unlike regular soft links, what it points to is a "dynamic" target depending on who accesses the symbolic link. The ideas that this filesystem represents are not limited just to handling electronic mail – that is only one particular application of this filesystem. See the sections on future work and alternative uses for other ideas.

Our key goals in designing HLFS were:

1. To provide every user with the ability to read mail from any host.
2. To ensure that all MTAs and UAs in use, as well as any other utilities which depend on the standard mail spool directory, face no problems from the change in the underlying filesystem.
3. To minimize the possibility of mail being lost or bouncing back to the sender.
4. To provide more privacy for users' mail files.

Since most sites provide access to users' home directories from any host, it made sense to store incoming mail there as well. That way, as long as users could log into a host and access their home directories via NFS, their mail would be accessible as well. This solved the first problem. Also, since users must login in order to read their mail, causing their home directories to be automounted, no extra directory mounts are required in order to begin reading mail.

The second problem was solved by making sure that the final access of the mail spool directory remains a "real" directory (not a read-only pseudo-filesystem provided by an automounter). All UAs access the spool directory for reading and writing the user's mail file and create lock files there.[9] That means that `/var/spool/mail` needs to be readable, writable, and searchable for UAs and MTAs so that lock files can be written and removed (also see the section on Lock files). For the purpose of writing the mail and lock files, a subdirectory inside the user's home directory is sufficient, since it is already owned by that user. Ensuring that users cannot access other users' files can easily be achieved by protecting this subdirectory.

In order not to change the behavior of programs like `comsat`[21,23], `from`, or `finger`, which are designed to read any user's mail, we implemented a special check for a designated group. If the effective *gid* of the process is the designated group, we assume that such a special program is executing, and `hlfsd` arranges to do the lookup not only of the real spool directory, but of the mailbox itself. See Table 2. Note that this method only supports read access without locking; however, this is sufficient for all programs that need to access other users' mailboxes. All that is required is to set these programs to be *setgid* to the designated group.

The third problem is solved by ensuring that all operations which might hang `hlfsd` are performed in the background, while still providing service in the parent or child process. Furthermore, if `hlfsd` is not running anymore, or if the user's home filesystem is full, mail gets delivered to an alternate

[9]Note that, in order to allow mail delivery to NFS-mounted mail spool directories, most vendors have modified the `/bin/mail` program to set its *uid* to that of the recipient when delivering mail. If a local delivery agent (LDA) on a system does not provide this behavior, the MTA must arrange to invoke it with the *uid* of the recipient – this can be done by a wrapper C program.

Conditions: *gid*=HLFS_GID for any *uid*.

Resolving component	Pathname left	Value if symbolic link
/	`var/mail/`*NAME*	
`var/`	`mail/`*NAME*	
`mail@`	`/mail/home/`*NAME*	`mail@` ⇒ `/mail/home`
/	`mail/home/`*NAME*	
`mail/`	`home/`*NAME*	
`home@`	*NAME*	`home@` ⇒ `.`
`./`	*NAME*	
NAME@		*NAME*@ ⇒ ~*NAME*`/.mailspool/`*NAME*
~*NAME*/	`.mailspool/`*NAME*	
`.mailspool/`	*NAME*	
NAME		

Table 2: Specially resolving `/var/mail/`*NAME* to ~*NAME*`/.mailspool/`*NAME*

directory (see the section entitled "Alternate Mail Spool Directories"). A `cron` job (running several times a day in our department), looks at the alternate directory, and attempts to resend messages in it to their rightful owners. All that is incurred is a delay in mail delivery, which, in most cases, is no longer than the length of time between consecutive invocations of the lost-mail remailing script.

Having a special mail-spool subdirectory owned and controlled by the user also addresses the last problem, that of privacy. Users can change the protection bits on their mailbox directory inside their home directory so that it is readable and searchable only by the owner. Any other program or user, unless running as the superuser on the same host,[10] would not be able to find out whether a user has new mail, how much of it there is, or when it was last read.

Implementation of `hlfsd`

We used a prototype NFS server, and implemented only the operations that were needed. We generated NFS stubs using `rpcgen`. The server was developed first under SunOS version 4.1.2. This server was incorporated into the amd source tree, and we used some of amd's sources as utility functions, since they are well-written to handle a variety of architectures and operating systems. (See the section on source code size, availability, and portability.)

The "Home-Link" File Service

This subsection includes technical details of the NFS operations and may be skipped. However, it provides an example of the design and implementation of a small special-purpose NFS server and may be of use to others.

HLFS is a read-only filesystem, and as such, all operations that require write access return the error code NFSERR_ROFS ("Read-Only Filesystem"): *setattr, write, create, remove, rename, link, unlink, symlink, mkdir,* and *rmdir*. Trivially implemented were the *null, root,* and *writecache* operations. We decided to have *statfs* return some value (all zeros in most cases). The *read* operation simply returns NFSERR_ACCES ("Permission Denied").

The remaining operations are the heart of this filesystem: *readdir, getattr, lookup,* and *readlink.*

Our server must distinguish between the directory and the link, so we assigned them different integers to serve as filehandles. Note that these need not be as complicated as the filehandles usually generated by NFS. They need only to be unique, and their value is meaningful only to the server.

[10]Or as the superuser elsewhere, if the filesystem is NFS-exported with `anon=0`.

The readdir Operation

Opening this directory returns the "." and ".." directories, and one symbolic link, home. Attempting to readdir on the symbolic link results in an NFSERR_NOTDIR. Anything else is a stale filehandle.

The getattr Operation

Getattr returns `r-xr-xr-x` for the "." and ".." directories. The link itself, named home by default, is protected as `rwxrwxrwx`. It does not matter for the link that it is world-writable. The modification and creation times for the link and directories are the startup time of the server. If the effective *gid* of the process is HLFS_GID, then some fixed valid attributes are returned. Any other filehandle given to `hlfsd` is considered stale and the NFSERR_STALE ("Stale Filehandle") result code is returned.

The lookup Operation

Obviously, we only allow looking up in the "." and ".." directories, both of which return the same values. Trying to lookup "in" the link results in an NFSERR_NOTDIR ("Not a Directory") error code. Any link not known to the server returns an NFSERR_NOENT ("No Such Entry") error, unless the *gid* of the requesting process is HLFS_GID and the name corresponds to a valid user. In this case the *username* for that user is used in the returned filehandle, allowing the readlink operation to return the correct link. Anything else is a stale filehandle.

The readlink Operation

This is the most important operation, the central point of this work. We get the *uid* number from the credentials sent with the RPC operation. We make sure that Unix Authentication or DES is used or else we return the NFSERR_PERM ("Not Owner") code.

If the *gid* of the accessing process is not HLFS_GID, the value we return for the symbolic link named home[11] is a string representing the home directory of the user whose *uid* we just found, concatenated with a fixed component name representing a subdirectory within it. We used a binary search on the lookup table to quickly get the right pathname. Different home directories for multiple password database entries with the same *uid* numbers may return any of the home directories. Only *uid* 0 is guaranteed to return "/" (see "Problems").

If the symbolic link is named home and the *gid* is HLFS_GID, we return a link to ".", which causes `hlfsd` to be used to resolve the next pathname component. This is designed to maintain functionality of programs such as `from`. If the symbolic link is not named home and the *gid* of the accessing process is HLFS_GID, we return a value pointing to the

[11]The name of the symbolic link is configurable.

user's mailbox file in their mail spool directory. To do this, we extract the *username* from the filehandle, which was returned by the lookup operation (see Table 2).

Trying to readlink on one of the two directories results in an NFSERR_ISDIR ("Is a directory") error. Anything else is a stale filehandle.

Execution Flow

At initialization time, `hlfsd` creates a UDP service, and forks a child. The child builds the *uid* lookup table, sets up signal handlers, and interval timers. The signal handlers are meant to reload the lookup table at expiration time of the interval timer, or when a SIGHUP is sent to the server (presumably by a superuser). A special cleanup handler is setup for SIGTERM, to ensure the server terminates cleanly. Then the `svc_run()` routine is invoked.

Meanwhile the parent waits for the child to initialize. When it does, the parent mounts the server on the mount point. Of utmost importance is to make sure the attribute cache is turned off. If the attribute cache is not turned off, successive accesses to `/mail/home` would return previously computed pathnames pointing to another user's mail, resulting in mail loss or misdelivery. If it is not possible to turn off the attribute cache, `hlfsd` will exit. However, the SA has the option to force `hlfsd` to continue running and set the attribute cache to as short an interval as possible (see "Problems"). At this point the parent terminates, leaving the child to run.

When an interval timer goes off (SIGALRM) or a SIGHUP is sent to the server, the server forks a child that continues serving, while the parent reloads the lookup table. When the parent is finished loading, it sends a SIGKILL to the child process, and resumes serving. When a SIGTERM is received, the server forks a child that continues serving, while it tries to unmount the filesystem. If and when that succeeds, both parent and child exit.

As mail service is very important, we wanted to make `hlfsd` as robust as possible. We could have designed it as another `amd` "filesystem type", but decided that a separate daemon provides better reliability and faster service. In general, we try to do as much as possible: we make sure filesystems are accessible and contain some disk space to have mail delivered there. Where directories are expected we make sure there are no files by these names; where symbolic links are expected, we make sure there are no real files or directories with the same name. Whenever possible, we create directories, with proper ownership and permissions. We even check that the mount point for `hlfsd` is world readable and executable, since if it isn't, `getwd("..")` might fail.

Alternate Mail Spool Directories

`Hlfsd` tries to ensure that the user's home directory is accessible. Periodically it also tests that it can be written into (see the section on disk space problems). If for any reason a failure occurs, `hlfsd` repoints the symbolic link for that user to an alternate local directory, which is presumably highly available. We use `/var/spool/alt_mail` in our environment. See Table 3.[12]

When `hlfsd` starts up, and before it mounts itself on top of the mount point, hiding anything that is underneath, `hlfsd` creates a fixed symbolic link to the alternate spool directory (if it does not exist already). This is done so that `/var/spool/mail` would not be a "dangling" symbolic link, and points to a real directory at all times, even after `hlfsd`

[12]In the conditions for Table 3, *~USER* is the home directory of the user with user-id *uid*.

Conditions: Any *uid*, *gid*≠HLFS_GID, and *~USER*`/.mailspool/` is <u>not</u> writable

Resolving component	Pathname left	Value if symbolic link
/	`var/mail/`*NAME*	
`var/`	`mail/`*NAME*	
`mail@`	`/mail/home/`*NAME*	`mail@` ⇒ `/mail/home`
/	`mail/home/`*NAME*	
`mail/`	`home/`*NAME*	
`home@`	*NAME*	`home@` ⇒ `/var/alt_mail`
/	`var/alt_mail/`*NAME*	
`var/`	`alt_mail/`*NAME*	
`alt_mail/`	*NAME*	
NAME		

Table 3: Resolving `/var/mail/`*NAME* to `/var/alt_mail/`*NAME*

terminates. When hlfsd runs, it hides this symbolic link, and provides our "dynamic" symbolic link. This trick at least provides us with an alternate place to deliver mail when things go wrong, rather than bounce or drop the mail.

A cron job on our systems checks the alternate mail spool directory several times a day. Any messages found there are resent to their rightful owners. The remailing script can be run as often as needed. Each invocation of the script deals only with newly lost mail since the previous invocation; the script locks and renames the lost mailbox file to a unique name, before parsing and remailing it.

Similar to amd, hlfsd can log debugging and various status information to a designated log file or using the syslog[22] facility. The SA may choose to watch these log files and facilities and be notified when serious problems occur such as a full filesystem.

Avoiding Hangs

As described in the section on execution flow, hlfsd forks a child at any point where we suspect that an operation might hang. If, for example, the home machine of the user is down, and the filesystem on a client is hard-mounted, hlfsd will hang until the remote server is back up. Performing these operations in the background provides added reliability, an idea taken from amd.

Disk Space Problems

Hlfsd checks if the user's home directory is full or they exceeded their quota. It attempts to create and then remove a simple nonzero-length file in the user's spool directory, with the effective *uid* set to that of the user. If that fails, it instead returns back the name of the alternate spool directory as the value of the home symbolic link. Otherwise mail might be dropped or bounce.

Any success or failure state is recorded in hlfsd. It is left there for a specified number of seconds, after which the entry "times out" and a new actual backgrounded lookup is required. Otherwise, the cached result is used and no expensive fork is required. This simple caching feature of hlfsd has greatly improved its performance and reliability. Also see the section on problems.

Lock Files

An alternative design for hlfsd is to have it mount on top of the mail spool directory directly, instead of having the mail spool directory be a symbolic link to another link (home) within the HLFS which points to a real subdirectory of the user's home. With some modifications to the server, we could have made all of the user's mailbox files point to the right place, but it suffered from serious drawbacks:

- The spool directory would no longer be a regular directory. It would have to be managed by hlfsd. This would require the implementation of more NFS operations.
- The user's spool file would not be a regular file, but a symbolic link to such. Some mail programs remove that file, not checking if it's a symbolic link. Therefore the symbolic link would be removed. We would have had to change the server so that removing the symbolic link would first follow it and remove the file it was pointing to. The same goes for all operations which require access to the user's mail spool file.
- The worst problem was that different UAs and MTAs use different methods for locking the mail file. Some of them create temporary files named ${USER}.lock, others use the mktemp library call to generate unique names. Our method avoids the need to figure out all the different methods used in locking mail files, and usage of temporary files.

An alternate way to avoid the need for lock files is to deliver mail one message per file using a different system such as with INN[19] and NNTP[10]; however, this would require modifications to all UAs and MTAs.

Source Code Size, Availability, and Portability

Hlfsd is less than 2500 lines of C code, including comments and white-spaces. However, it makes use of almost 4000 lines of code from the amd distribution itself.

Hlfsd is available in source form as part of a special distribution of amd. It can be retrieved via anonymous ftp from ftp.cs.columbia.edu in the directory /pub/amd.

Hlfsd is built as part of the special distribution of amd available from our ftp server. It is almost as portable as amd is. It is only the lack of access to certain machines that stopped us from porting hlfsd to the numerous platforms amd runs on. At the writing of this paper, hlfsd has been successfully ported and running on SunOS 4.1.3, HP-UX 9.0.1, and Solaris 2.2. Those represent the 3 main system types amd runs on and span most Unix flavors: a BSD-style system, an SVR4-BSD hybrid, and a system very close to SVR4, respectively.

Evaluation

This system is implemented and has been in use on a number of machines for more than a year now. For the first nine months hlfsd was in experimental use. We have since deployed it on most production machines in our department, spanning over a 100 hosts and 3 different architectures.

The goal of this work is to expand the accessibility of electronic mail, improve overall reliability and stability of this vital service, while at the same time maintain the sanity of our SAs (yours truly included). For this to really work, it must have:

1. Very high availability.
2. Little overhead.
3. Little hassle for users and administrators as the system is being used or installed for the first time.

Performance

We have carried out some measurements to quantify the above requirements and more. The tests were performed on a Sun SPARCstation-2 running SunOS 4.1.3.

Accessing a local spool file via `stat` normally takes 180 msec without `hlfsd`. If `hlfsd` is running and has the user's entry already cached, it takes 60 msec more to access the file. This overhead is attributed to the fact that the kernel has to access a user-level NFS server, making several context switches.

If the entry is not cached, `hlfsd` forks a child to perform operations which may cause it to hang. The overhead of that `fork` and other checks is an additional 70 msec (or 130 msec over the regular lookup not using `hlfsd`). However, this overhead is incurred only once in 5 minutes, because the result of each check is cached for that long by default.

The above times are somewhat significant, but not by much, considering the use of a user-level file-server. (By comparison, in our environment it takes about 0.5 seconds to access a new filesystem using amd.) Given the benefits of `hlfsd`, we feel that a minimal access slowdown is a small price to pay. In practice, over 12 months of usage we have noticed no visible degradation of service.[13]

The internal data structures (tables and caches) require 50 bytes per user on the system. In our environment (750 users), that totals about 37KB – rather insignificant given that workstations these days come installed with at least 16-32MB of RAM.

Remailing Lost Mail

The `hlfsd` distribution contains a `perl`[26] script called `lostaltmail`. Remailing a single message with a body size of 1KB takes an average of 1.2 seconds (total time). In our department, resending an average mailbox file takes about 20 seconds.

Initially we ran the script once a day, but found having to wait up to 24 hours for lost mail redelivery too long. We then experimented with running `lostaltmail` once an hour. However, we found that frequency too fast. The most likely situation in which `hlfsd` will repoint its symbolic link to the alternate spool directory is when the user's filesystem is full. A full filesystem is a persistent situation that in most cases takes some time to get fixed, as it requires human intervention. If the situation that causes `hlfsd` to use the alternate spool directory is likely to persist, running the `lostaltmail` script will consume unnecessary resources, only to redeliver the mail back to the alternate spool directory. We finally settled on running `lostaltmail` between 6 and 12 times a day. Depending on the amount of lost mail expected, the script could be run more or less often.

Reliability

We have simulated worst-case scenarios by filling up a user filesystem and letting `hlfsd` decide to redirect mail to the alternate spool directory. At this point we filled up that filesystem as well. `Hlfsd` kept on pointing to the alternate spool directory during the cached entry interval, but we observed no mail lost. Instead, the sending side detected that the filesystem was full, and kept the message in the remote (private) spool directory. This is the default behavior `sendmail`[1] provides.

`Hlfsd` does not introduce any new problems; that is, if a filesystem is completely full, whatever behavior your current LDA provides is maintained. Since `hlfsd` uses both the user's filesystem and an alternate spool directory, it actually increases the availability of mail services, by "virtually" increasing the disk space available for mail spooling.

Once space has been freed in the user's filesystem, and the cached entry expired, `hlfsd` pointed its symbolic link back to the user's home directory. The next time the remailing script ran, all "lost" mail got resent to its owners.

Since the installation of `hlfsd` in our production environment, we have seen a few cases of lost mail being resent, mostly due to full filesystems. We know of no case where mail was completely lost.

Installation

Since `hlfsd` was written by SAs for other SAs, we have provided it with several command-line options to use at startup time, enabling `hlfsd` to be tailored for a particular environment. Needless to say, a man page is provided, as well as complete source code. Furthermore, we included a few scripts written in `sh` and `perl` which we use in our environment to re/start `hlfsd`, test for possible configuration anomalies, and resend "lost" mail.

The most significant work that SAs need to do is identify programs that need to access mailbox files of other users, and "*setgid*" them to HLFS_GID. In our environment we had to do that for `comsat`, `from`, `finger` and a few others. Our environment uses the `rdist`[5] automatic software distribution program, and thus these changes were required only in one place – the top of our `rdist` tree.

[13]The SAs group felt so convinced that `hlfsd` was working well, that we were the first to use it on *our* home machines.

Problems

There are a few problems, some of which cannot be easily resolved:

- Some programs need to be *setgid* to the special HLFS_GID group. There is no easy way to locate them other than knowing ahead of time what they do. Note that if the programs are not *setgid*, the only consequence is that these programs are unable to find mailboxes. However, with other methods, if $MAIL is not used, mail is not delivered.
- It is possible that the status of a home directory access will change during the time that `hlfsd` caches this information. Picking a smaller cache expiration time can alleviate this problem, but it increases the resources taken by `hlfsd` and slows down access to mail. It is left for the individual SAs to change this default value.
- Any logins with the same *uid* and a different home directory may have mail delivered or read from any of their home directory pathnames. `Hlfsd` stores pathnames in an internal hash table keyed by the *uid*; therefore, it is undefined which pathname is returned in the case of multiple users with the same *uid* and different home directories. We provide a script which checks for this situation and warns the SAs.
- On systems that cannot turn off the NFS attribute cache, the kernel might return the same symbolic link name for two different users who access the spool directory consecutively, possibly resulting in mail getting delivered to the wrong mailbox! On these systems, `hlfsd` will not run unless started with a special option. In that case it will set the attribute cache value to the shortest possible interval, but it may not be sufficient.

Related Work

The idea of dynamic or variable pathname components is not new. HP-UX does this with context-dependent files[8], and Mach with the "`@SYS`" variables[4]. Both of these implementations support replacement of pathname components by kernel variables. Apollo's DOMAIN/OS supported a more sophisticated system where arbitrary user environment variables could be referenced in pathnames[11,12]. On the issue of having a user's home files and mailbox file reside in one location, Plan 9's *attach* operator can be used to unify several file servers into one user name space[14].

What is new about our idea is that we do not require any change to any part of the filesystem implementation in the kernel. All that is required are RPC and NFS, making the system much more widely applicable.

Though at first it may appear that amd can do what hlfsd does, it can't. Amd cannot return different pathnames as a value of a symbolic link depending on who accessed it. See the earlier section on Distributed Mail Spool Service for more details on various ways in which amd cannot help the way `hlfsd` can.

Mail-reading Servers

The future of mail reading and sending may be similar to that used by the NNTP protocol used for managing NetNews[10,19]. That is, a special-purpose server which provides network connections for reading and writing mail remotely.

Several such programs exist, most notably IMAP[16] and POP[17]. However, use of these servers is limited at this time because most MTAs and UAs have not been converted to use them, or they require special environments (the *Andrew Message Delivery System*[18] requires AFS). Porting those applications for most popular environments is not going to be an easy task. Nevertheless, the benefits of such services over that of `hlfsd` would include faster and more reliable service, plus greatly expanded functionality (possibly providing threads information for threaded mail readers).

Conclusion

We have described the benefits of delivering mail to users' home directories, the traditional ways to do that and why we think they are inadequate, and the design, implementation, performance, and convenience of our alternative.

The main contribution of our work is the idea mail can be reliably delivered to user's home directories for easy access with very little overhead, user hassle, or the need for extensive intervention on the part of SAs.

A working prototype version of `hlfsd` was written in one weekend. However, the ideas represented in the work span several years of experience in network programming (especially RPC), NFS, amd, and mail systems.

Future Work

It would be possible to integrate some of `hlfsd`'s functionality into amd, by providing special keywords like `${home}`, `${user}`, and `${group}` for use in amd's maps.

We plan on making sure `hlfsd` is as portable as amd is, and improving its performance as much as possible. An RPC interface for querying `hlfsd`'s status is needed as well.

Alternative Uses

`Hlfsd`'s primary use is that of a mail-spool redirector. However, it can be used to perform other tasks. All it takes are the right command-line options:

- `Hlfsd` can manage the `/var/tmp` directory. Thus every user who uses `/var/tmp` would actually be using a subdirectory within their own home directory, rather than taking from system-wide resources.
- Other types of user-specific files which get spooled to a particular host, such as *Secret Mail*[24] or electronic faxes can also be redirected for spooling into home directories.

Acknowledgments

Special thanks go to Daniel Duchamp for his invaluable comments on the paper, to James Tanis who wrote the remailing script and provided useful feedback, and to all members of the technical staff who helped in stress-testing `hlfsd`. We would like to thank the many members of the amd mailing list (`amd-workers@acl.lanl.gov`) for providing valuable discussion on the subject.

This work was supported in part by a National Science Foundation CISE Institutional Infrastructure grant, number CDA-90-24735.

As `hlfsd` uses parts of the amd distribution, it is distributed under the same restrictions and licenses that amd is.

Author Information

Erez Zadok is an MS candidate and full-time Staff Associate in the Computer Science Department at Columbia University. His primary interests include operating systems, file systems, and ways to ease system administration tasks. In May 1991, he received his B.S. in Computer Science from Columbia's School of Engineering and Applied Science. Erez came to the United States seven years ago and has lived in New York "Sin" City ever since. In his rare free time, Erez is an amateur photographer, science fiction devotee, and rock-n-roll fan. Mailing address: 500 West 120th Street, Columbia University, New York, NY 10027. Email address: ezk@cs.columbia.edu.

Alexander Dupuy has been a Senior Research Staff Associate for the Distributed Computing and Communications Lab in the Computer Science Department at Columbia University for the last 7 years. He has recently taken a position at System Management ARTS, a small startup company developing network and systems management technology. A native born and bred New Yorker, he insists that working in the suburbs is not the first step towards living there. Mailing address: System Management ARTS, 199 Main Street, Suite 900, White Plains, NY 10601. Email address: dupuy@smarts.com.

References

[1] E. Allman. SENDMAIL – An Internetwork Mail Router. In *UNIX System Manager's Manual.* University of California, Berkeley, 1986.

[2] F. C. Baran. MW: Mail-Watch. An unpublished manual page, Academic Systems Group, Columbia University Center for Computing Activities, 1987.

[3] B. Callaghan and T. Lyon. The Automounter. In *Proc. 1989 Winter USENIX Conf.*, pp. 43-51, January 1989.

[4] M. N. Condict. Configuring and Building Mach 3.0. OSF Research Institute, Grenoble, France. Unpublished notes available via ftp from `mach.cs.cmu.edu` in the file `doc/notes/kernel_build.doc`.

[5] M. A. Cooper. Overhauling Rdist for the '90s. *Large Installation System Administrators Workshop Proceedings*, pp. 1-8, USENIX, Long Beach, CA, October 19-June 23, 1992.

[6] A. J. Findlay. The Home-Directory Mail System. In *EUUG News*, Autumn 1988.

[7] J. Fulton. MIT X Consortium. X11R5 Reference Manual Pages, Section 1: "xbiff(1)", 1988.

[8] Hewlett-Packard Company. HP-UX Release 9.0 Reference Manual, Section 4: "cdf(4)", August 1992.

[9] IBM Corp. AIX Commands Reference, Volume 1, "bellmail(1)", pp. 1-84–1-87, December 1989.

[10] B. Kantor and P. Lapsley. Network News Transfer Protocol. RFC 977, February 1986; 27 p.

[11] P. J. Leach, P. H. Levine, B. P. Douros, J. A. Hamilton, D. L. Nelson, and B. L. Stumpf. *The Architecture of an Integrated Local Network.* In *IEEE Journal on Selected Areas in Communications,* **SAC-1** (5), pp. 842-856, November 1983.

[12] P. H. Levine. *The Apollo DOMAIN Distributed File System.* In *NATO ASI Series: Theory and Practice of Distributed Operating Systems,* Y. Paker, J-P. Banatre, M. Bozyiğit, pp. 241-260, editors, Springer-Verlag, 1987.

[13] J. S. Pendry and N. Williams. *Amd – The 4.4 BSD Automounter.* Imperial College of Science Technology and Medicine, London. March 1991.

[14] R. Pike, D. Presotto, K. Thompson, and H. Trickey. *Plan 9 from Bell Labs.* In *Proceedings of the Summer 1990 UKUUG Conf.*, London, July, 1990, pp. 1-9.

[15] J. B. Postel. Simple Mail Transfer Protocol. RFC 821, August 1982; 68 p.

[16] J. Rice. Interactive Mail Access Protocol. RFC 1203, February 1991; 49 p.

[17] M. Rose. Post Office Protocol. RFC 1225, May 1991; 16 p.

[18] J. Roseneberg, C. F. Everhart, and N. S. Borenstein. *An Overview of the Andrew Message System.* In *Proceedings of the ACM SIGCOMM '87 Workshop*, Stowe, Vermont, August 11-13, 1987, pp. 99-108.

[19] R. Salz. *InterNetNews: Usenet transport for Internet sites.* In *Proc. 1992 Summer USENIX Conf.*, pages 93-98, June 1992.

[20] R. Sandberg, et al. Design and Implementation of the Sun Network Filesystem. In *Proc. 1985 Summer Usenix Conf.*, pages 119-130, June 1985.

[21] Sun Microsystems, Inc. SunOS Reference Manual, Volume I, Section 1: "biff(1)", September 9, 1987.

[22] Sun Microsystems, Inc. SunOS Reference Manual, Volume II, Section 3: "syslog(3)", September 9, 1987.

[23] Sun Microsystems, Inc. SunOS Reference Manual, Volume I, Section 1: "comsat(8c)", and "in.comsat(8c)", September 9, 1987.

[24] Sun Microsystems, Inc. SunOS Reference Manual, Volume I, Section 1: "xsend(1)", "xget(1)", and "enroll(1)", September 9, 1987.

[25] Sun Microsystems, Inc. NFS: Network File System Protocol specification. RFC 1094, 1989 March; 27 p.

[26] L. Wall and R. L. Schwartz. *Programming Perl.* O'Reilly & Associates, Inc., Sebastopol, CA (1991).

[27] R. S. Zachariassen. *ZMOG: The ZMailer Operations Guide.* Available via ftp as part of the ZMailer distribution from `ftp.uu.net` in the `networking/mail/zmailer` directory.

NAME

hlfsd – home-link file system daemon

SYNOPSIS

hlfsd [**–Cfhnpv**] [**–a** *alt_dir*] [**–c** *cache-interval*] [**–g** *group*] [**–i** *reload-interval*] [**–l** *logfile*] [**–o** *mount-options*] [**–x** *log-options*] [**–D** *debug-options*] [*linkname* [*subdir*]]

DESCRIPTION

Hlfsd is a daemon which implements a filesystem containing a symbolic link to subdirectory within a user's home directory, depending on the user which accessed that link. It was primarily designed to redirect incoming mail to users' home directories, so that it can read from anywhere.

Hlfsd operates by mounting itself as an NFS server for the directory containing *linkname*, which defaults to **/hlfs/home**. Lookups within that directory are handled by **hlfsd**, which uses the password map to determine how to resolve the lookup. The directory will be created if it doesn't already exist. The symbolic link will be to the accessing user's home directory, with *subdir* appended to it. If not specified, *subdir* defaults to **.hlfsdir**. This directory will also be created if it does not already exist.

A SIGTERM sent to **hlfsd** will cause it to shutdown. A SIGHUP will flush the internal caches, and reload the password map.

OPTIONS

–a *alt_dir*
Alternate directory. The name of the directory to which the symbolic link returned by **hlfsd** will point, if it cannot access the home directory of the user. This defaults to **/var/hlfs**. This directory will be created if it doesn't exist. It is expected that either users will read these files, or the system administrators will run a script to resend this "lost mail" to its owner.

–C Force **hlfsd** to run on systems that cannot turn off the NFS attribute-cache. Use of this option on those systems is discouraged, as it may result in loss or misdelivery of mail. The option is ignored on systems that can turn off the attribute-cache.

–c *cache-interval*
Caching interval. **Hlfsd** will cache the validity of home directories for this interval, in seconds. Entries which have been verified within the last *cache-interval* seconds will not be verified again, since the operation could be expensive, and the entries are most likely still valid. After the interval has expired, **hlfsd** will re-verify the validity of the user's home directory, and reset the cache time-counter. The default value for *cache-interval* is 300 seconds (5 minutes).

–f Force fast startup. This option tells **hlfsd** to skip startup-time consistency checks such as existence of mount directory, alternate spool directory, symlink to be hidden under the mount directory, their permissions and validity.

–g *group*
Set the special group HLFS_GID to *group*. Programs such as **from** or **comsat**, which access the mailboxes of other users) must be setgid HLFS_GID to work properly. The default group is "hlfs". If no group is provided, and there is no group "hlfs", this feature is disabled.

–h Help. Print a brief help message, and exit.

–i *reload-interval*
Map-reloading interval. Each *reload-interval* seconds, **hlfsd** will reload the password map. **Hlfsd** needs the password map for the UIDs and home directory pathnames. **Hlfsd** schedules a SIGALRM to reload the password maps. A SIGHUP sent to **hlfsd** will force it to reload the maps immediately. The default value for *reload-interval* is 900 seconds (15 minutes.)

–l *logfile*
Specify a log file to which **hlfsd** will record events. If *logfile* is the string **syslog** then the log messages will be sent to the system log daemon by *syslog*(3), using the LOG_DAEMON

facility. This is also the default.

–n No verify. **Hlfsd** will not verify the validity of the symbolic link it will be returning, or that the user's home directory contains sufficient disk-space for spooling. This can speed up **hlfsd** at the cost of possibly returning symbolic links to home directories which are not currently accessible or are full. By default, **hlfsd** validates the symbolic-link in the background. The **–n** option overrides the meaning of the **–c** option, since no caching is necessary.

–o *mount-options*
Mount options. Mount options which **hlfsd** will use to mount itself on top of *dirname.* By default, *mount-options* is set to "ro", unless M_CACHE is defined, in which case it is set to "ro,nocache".

–p Print PID. Outputs the process-id of **hlfsd** to standard output where it can be saved into a file.

–v Version. Displays version information to standard error.

–x *log-options*
Specify run-time logging options. The options are a comma separated list chosen from: fatal, error, user, warn, info, map, stats, all.

–D *log-options*
Select from a variety of debugging options. Prefixing an option with the string **no** reverses the effect of that option. Options are cumulative. The most useful option is **all.** Since this option is only used for debugging other options are not documented here. A fuller description is available in the program source. A SIGUSR1 sent to **hlfsd** will cause it to dump its internal password map to the file **/tmp/hlfsdump**.

FILES

/hlfs directory under which **hlfsd** mounts itself and manages the symbolic link **home.**

.hlfsdir
default sub-directory in the user's home directory, to which the **home** symbolic link returned by **hlfsd** points.

/var/hlfs
directory to which **home** symbolic link returned by **hlfsd** points if it is unable to verify the that user's home directory is accessible.

SEE ALSO

amd(8), **automount**(8), **cron(8), getgrent**(3), **getpwent**(3), **mail(1), mount**(8), **mtab**(5), **passwd**(5), **sendmail**(8), **umount**(8).

AUTHORS

Erez Zadok <ezk@cs.columbia.edu>, Computer Science Department, Columbia University, New York City, New York, USA, and Alexander Dupuy <dupuy@smarts.com>, System Management ARTS, White Plains, New York, USA.

The USENIX Association

The UNIX and Advanced Computing Systems Professional and Technical Association

The USENIX Association is a not-for-profit membership organization of those individuals and institutions with an interest in UNIX and UNIX-like systems and, by extension, C++, X windows, and other programming tools. It is dedicated to:

- sharing ideas and experience relevant to UNIX or UNIX inspired and advanced computing systems,
- fostering innovation and communicating both research and technological developments,
- providing a neutral forum for the exercise of critical thought and airing of technical issues.

Founded in 1975, USENIX sponsors twice yearly general conferences accompanied by vendor displays and frequent single-topic conferences and symposia. USENIX publishes proceedings of its meetings, the bi-monthly newsletter ;login:, the refereed technical quarterly Computing Systems. (published with the University of California Press), and is expanding its publishing role in cooperation with MIT Press with a book series on advanced computing systems. The Association actively participates in various ANSI, IEEE and ISO standards efforts with a paid representative attending selected meetings News of standards efforts and reports of many meetings are reported in ;login:.

SAGE, the Systems Administrators' Guild

USENIX has recently launched its first Special Technical Groups (STGs), the Systems Administrators' Guild (SAGE). SAGE is devoted to the advancement of systems administration as a profession. It will recruit talented individuals to the profession, develop guidelines for the education of members of the profession, establish standards of professional excellence and provide recognition for those who attain them, and promote work that advances the state of the art and propagates knowledge of good practice in the profession.

USENIX and SAGE will work together to publish technical information and sponsor conferences, symposia , tutorials and local groups in the field of systems administration. Currently USENIX and SAGE jointly sponsor the annual Systems Administration Conference and they, together with FedUNIX, are sponsoring the 1993 World Conference on Tools and Techniques for Systems Administration, Networking and Security (SANS-II). SAGE News and other items of interest to systems administrators are found in each issue of the USENIX newsletter ;login:.

There are four classes of membership in the Association, differentiated primarily by the fees paid and services provided.

USENIX Association services include:

- Subscription to login:, a bi-monthly newsletter;
- Computing Systems, a refereed technical quarterly;
- Discounts on various UNIX and technical publications for purchase;
- Technical conference and tutorial program twice a year and single-topic symposia periodically;
- A discount on technical conference and workshop registration fees;
- The right to vote on matters affecting the Association, its bylaws, election of its directors and officers.
- Right to join Special Technical Groups such as SAGE

The supporting members of the USENIX Association are:

ASANTÉ Technologies, Inc.	OTA Limited Partnership
ANDATACO	Quality Micro Systems
Frame Technology, Inc.	Sybase, Inc.
Matsushita Electrical Industrial Co., Ltd.	UNIX System Laboratories, Inc.
Network Computing Devices, Inc.	UUNET Technologies, Inc.

For further information about membership, conferences or publications, contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Telephone: 510/528-8649
Email: office@usenix.org
FAX: 510/548-5738

ISBN 1-880446-56-1